Inteligencia Artificial

Búsqueda en espacio de estados

Edgar Andrade, Ph.D.

Matemáticas Aplicadas y Ciencias de la computación

Última revisión: Agosto de 2023

MACC
Matemáticas Aplicadas y
Ciencias de la Computación

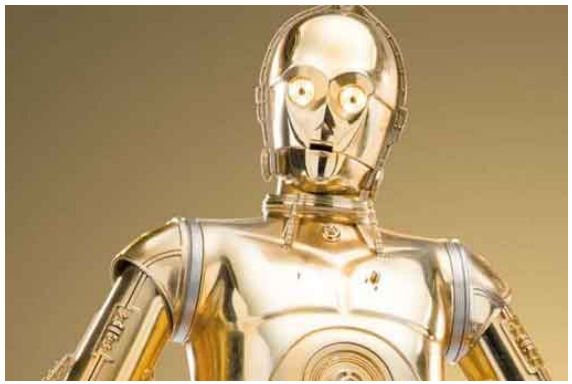
Contenido

MACC
Matemáticas Aplicadas y Ciencias de la Computación

Contenido

Ambientes de tarea

La IA no es un alma perdida en un universo vacío. Más bien, el objetivo aquí es construir un agente que percibe y actua en un entorno para atender una tarea concreta.

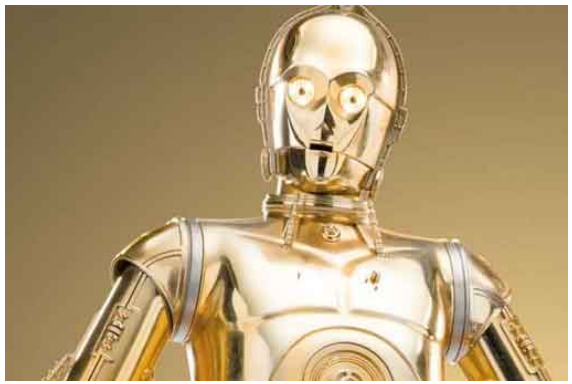


	Sensores	Actuadores	Entorno	Medida de desempeño
Ajedrez				

MACC
Matemáticas Aplicadas y Ciencias de la Computación

Ambientes de tarea

La IA no es un alma perdida en un universo vacío. Más bien, el objetivo aquí es construir un agente que percibe y actua en un entorno para atender una tarea concreta.

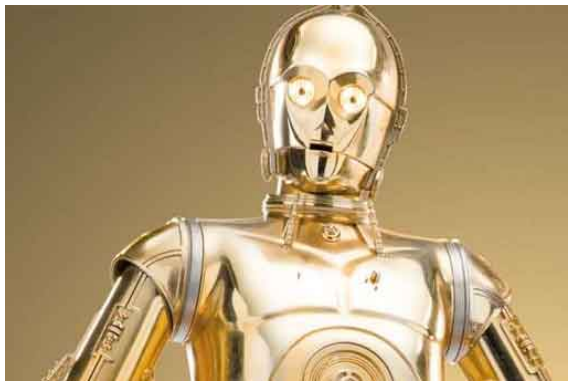


	Sensores	Actuadores	Entorno	Medida de desempeño
Ajedrez	Percepción del Tablero			

MACC
Matemáticas Aplicadas y Ciencias de la Computación

Ambientes de tarea

La IA no es un alma perdida en un universo vacío. Más bien, el objetivo aquí es construir un agente que percibe y actua en un entorno para atender una tarea concreta.

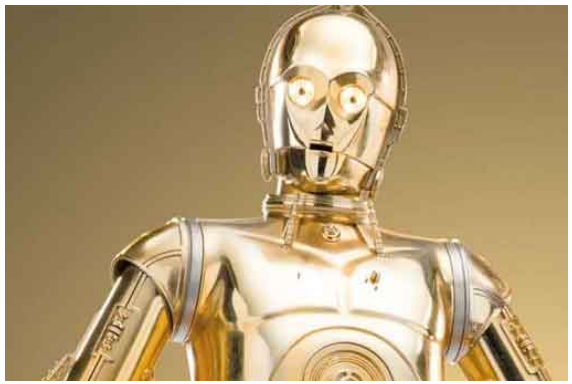


	Sensores	Actuadores	Entorno	Medida de desempeño
Ajedrez	Percepción del Tablero	Movimiento de las fichas		

MACC
Matemáticas Aplicadas y Ciencias de la Computación

Ambientes de tarea

La IA no es un alma perdida en un universo vacío. Más bien, el objetivo aquí es construir un agente que percibe y actua en un entorno para atender una tarea concreta.

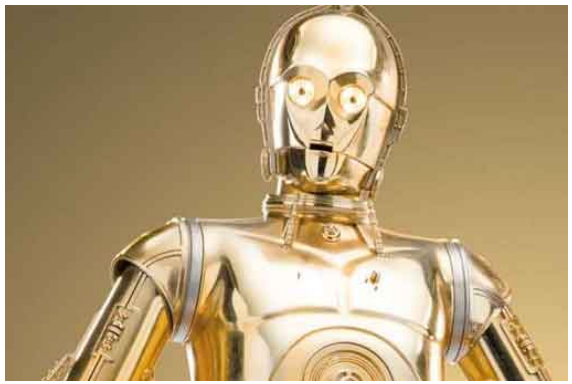


	Sensores	Actuadores	Entorno	Medida de desempeño
Ajedrez	Percepción del Tablero	Movimiento de las fichas	Tablero	

MACC
Matemáticas Aplicadas y Ciencias de la Computación

Ambientes de tarea

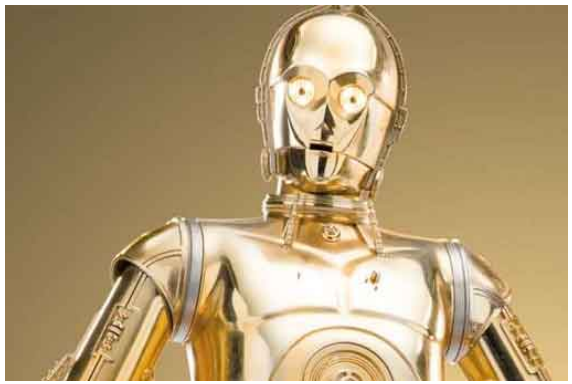
La IA no es un alma perdida en un universo vacío. Más bien, el objetivo aquí es construir un agente que percibe y actua en un entorno para atender una tarea concreta.



	Sensores	Actuadores	Entorno	Medida de desempeño
Ajedrez	Percepción del Tablero	Movimiento de las fichas	Tablero	Ganar> Empatar> Perder -tiempo

Ambientes de tarea

La IA no es un alma perdida en un universo vacío. Más bien, el objetivo aquí es construir un agente que percibe y actua en un entorno para atender una tarea concreta.



	Sensores	Actuadores	Entorno	Medida de desempeño
Ajedrez	Percepción del Tablero	Movimiento de las fichas	Tablero	Ganar> Empatar> Perder -tiempo
C3PO				

Propiedades (1/4)

Fully observable vs. **Partially observable**

El agente puede percibir todo el entorno

El agente NO puede percibir todo el entorno

Propiedades (1/4)

Fully observable vs. **Partially observable**

El agente puede percibir todo el entorno

El agente NO puede percibir todo el entorno

Single agent vs. **Multiagent**

El agente está solo en el entorno

Varios agentes interactúan en el entorno

Propiedades (2/4)

Deterministic	vs.	**Stochastic**
El entorno tiene una dinámica completamente determinada		La dinámica del entorno exhibe aleatoriedad

Propiedades (2/4)

Deterministic vs. **Stochastic**

El entorno tiene una dinámica completamente determinada

La dinámica del entorno exhibe aleatoriedad

Episodic vs. **Sequential**

La solución se obtiene mediante una sola acción

Se requiere una secuencia de acciones para lograr el objetivo

MACC
Matemáticas Aplicadas y Ciencias de la Computación

Propiedades (3/4)

Static vs. **Dynamic**

El entorno no cambia si el agente no lo cambia

El entorno tiene una dinámica independiente del agente

Propiedades (3/4)

Static vs. **Dynamic**

El entorno no cambia si el agente no lo cambia

El entorno tiene una dinámica independiente del agente

Discreto vs. **Continuo**

El entorno está constituido por bloques distintos

El entorno está compuesto por un espacio continuo

MACC
Matemáticas Aplicadas y Ciencias de la Computación

Propiedades (4/4)

Known vs. **Unknown**

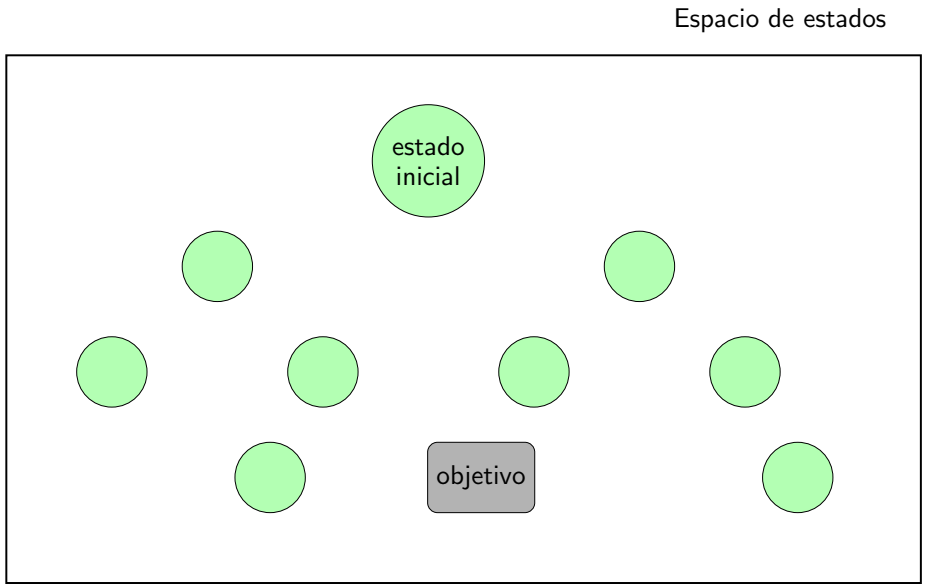
El agente conoce las reglas del entorno

El agente desconoce en parte cómo funciona el entorno

Contenido

MACC
Matemáticas Aplicadas y Ciencias de la Computación

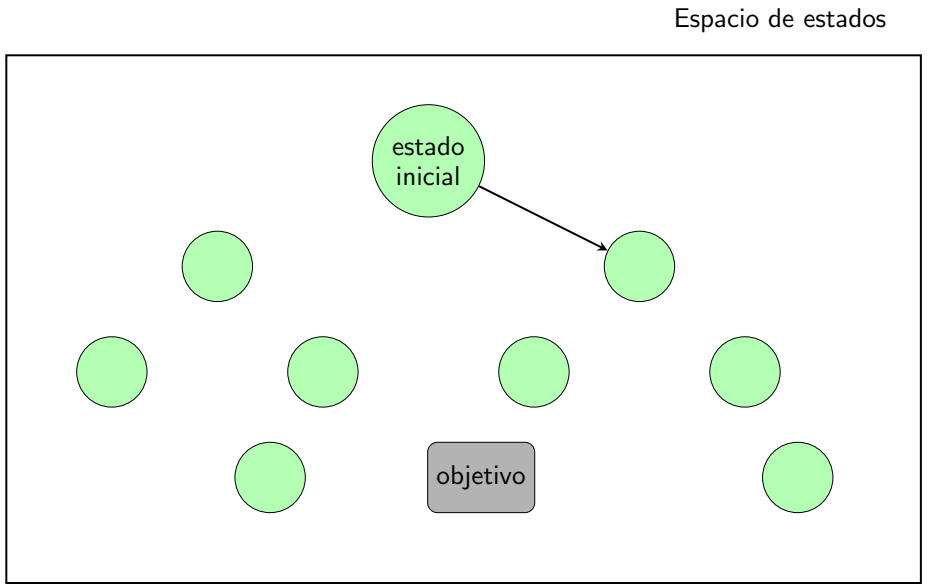
Idea general



- Estado inicial
- Acciones aplicables
- Transición
- Test objetivo

MACC
Matemáticas Aplicadas y Ciencias de la Computación

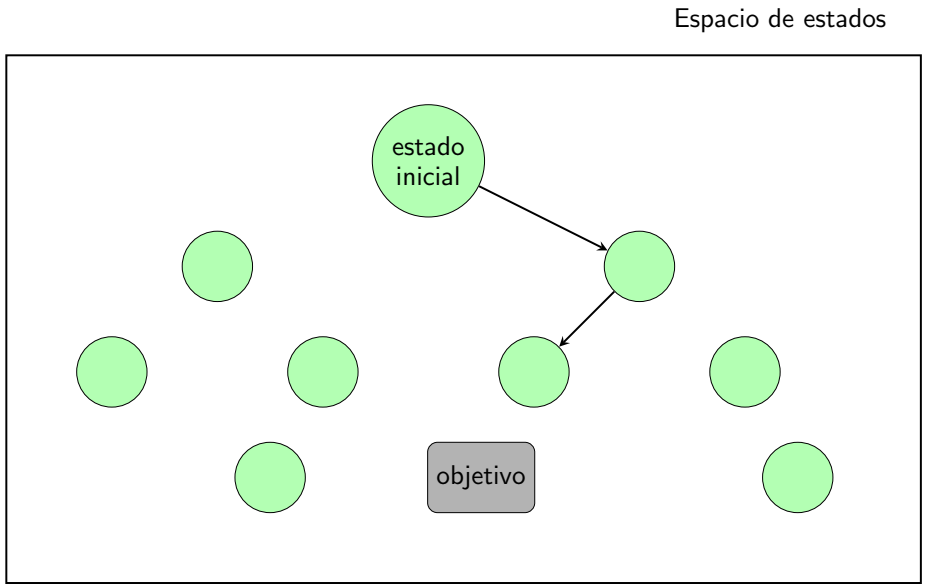
Idea general



- Estado inicial
- Acciones aplicables
- Transición
- Test objetivo

MACC
Matemáticas Aplicadas y Ciencias de la Computación

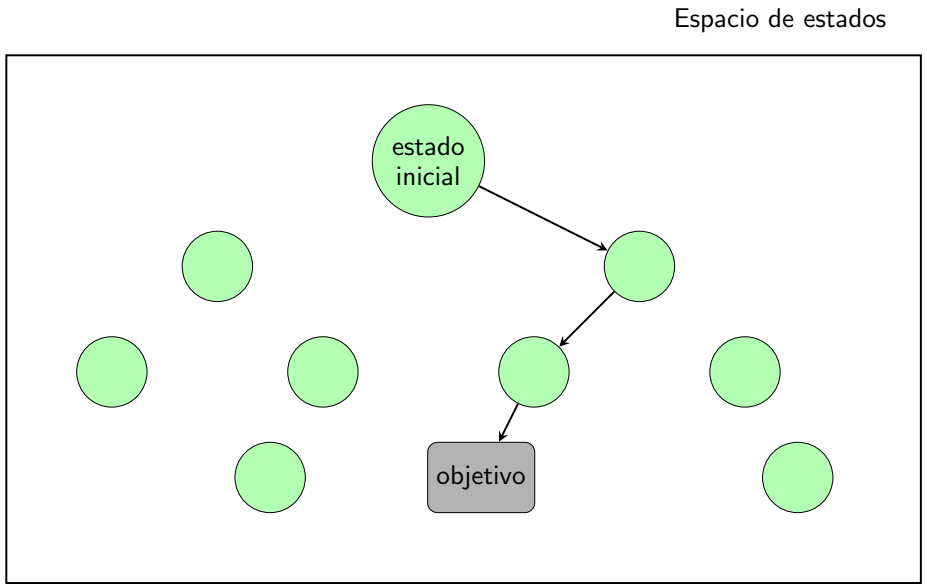
Idea general



- Estado inicial
- Acciones aplicables
- Transición
- Test objetivo

MACC
Matemáticas Aplicadas y Ciencias de la Computación

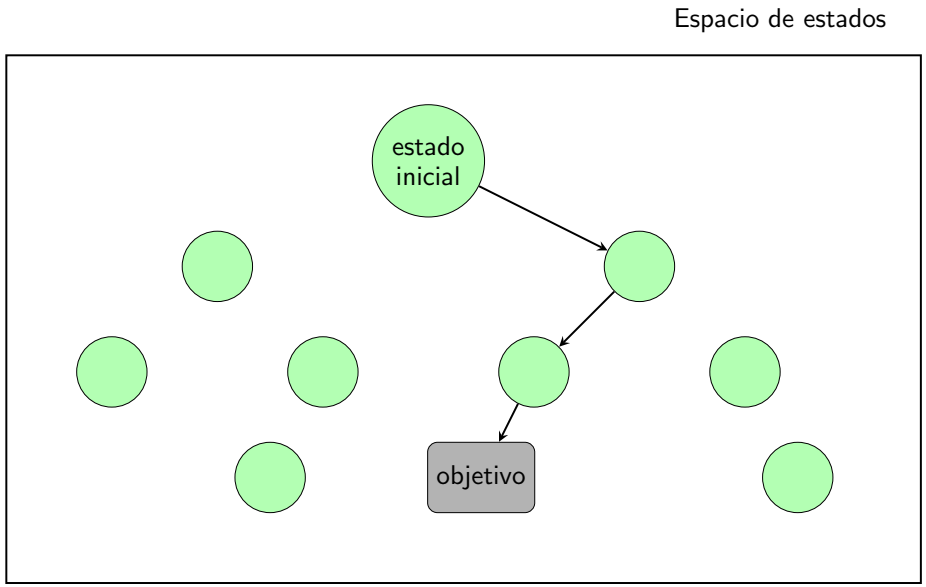
Idea general



- Estado inicial
- Acciones aplicables
- Transición
- Test objetivo

MACC
Matemáticas Aplicadas y Ciencias de la Computación

Idea general

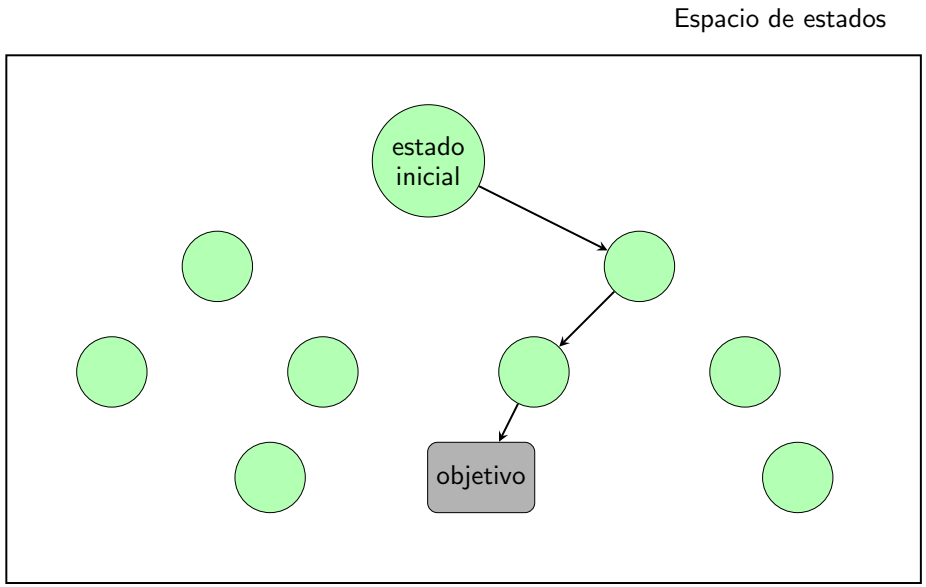


- Estado inicial
- Acciones aplicables
- Transición
- Test objetivo

☞ ¿Cómo hacer una búsqueda sistemática del espacio de estados?

MACC
Matemáticas Aplicadas y Ciencias de la Computación

Idea general



- Estado inicial
- Acciones aplicables
- Transición
- Test objetivo

☞ ¿Cómo hacer una búsqueda sistemática del espacio de estados?

☞ Búsqueda a ciegas vs. Búsqueda heurística

MACC
Matemáticas Aplicadas y Ciencias de la Computación

Contenido

Tipos de listas — FIFO y LIFO

First In First Out

FIFO =

Tipos de listas — FIFO y LIFO

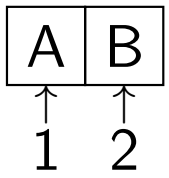
First In First Out

$$\text{FIFO} = \underset{\uparrow}{\boxed{\text{A}}}$$
$$1$$

ADD(FIFO, A)

Tipos de listas — FIFO y LIFO

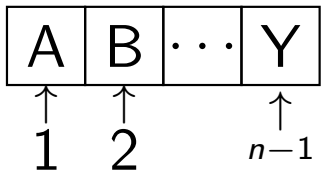
First In First Out

FIFO = 

ADD(FIFO, B)

Tipos de listas — FIFO y LIFO

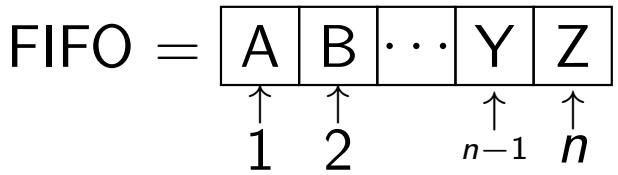
First In First Out

FIFO = 

ADD(FIFO, Y)

Tipos de listas — FIFO y LIFO

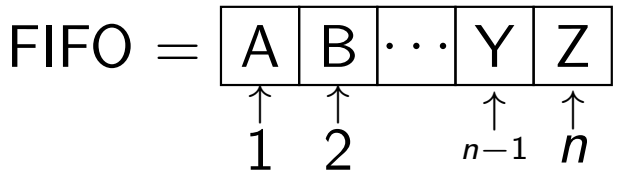
First In First Out



ADD(FIFO, Z)

Tipos de listas — FIFO y LIFO

First In First Out



$s \leftarrow \text{POP}(\text{FIFO})$

Tipos de listas — FIFO y LIFO

First In First Out

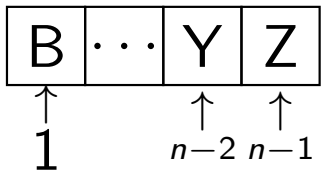
$$\text{FIFO} = \boxed{\text{B}}\boxed{\cdots}\boxed{\text{Y}}\boxed{\text{Z}}$$

$$\uparrow_{1} \qquad \uparrow_{n-2} \; \uparrow_{n-1}$$

$s \leftarrow$ A

Tipos de listas — FIFO y LIFO

First In First Out

FIFO = 

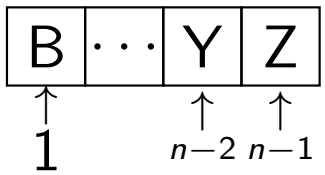
$s \leftarrow$ A

Last In First Out

LIFO =

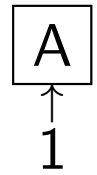
Tipos de listas — FIFO y LIFO

First In First Out

FIFO = 

$s \leftarrow$ A

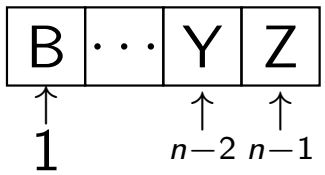
Last In First Out

LIFO = 

ADD(FIFO, A)

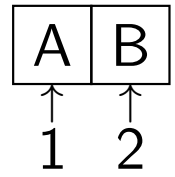
Tipos de listas — FIFO y LIFO

First In First Out

FIFO = 

$s \leftarrow$ A

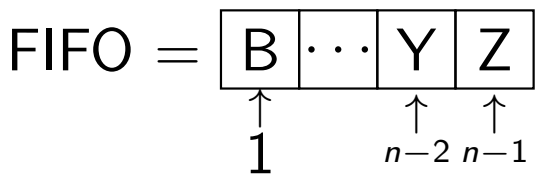
Last In First Out

LIFO = 

ADD(FIFO, B)

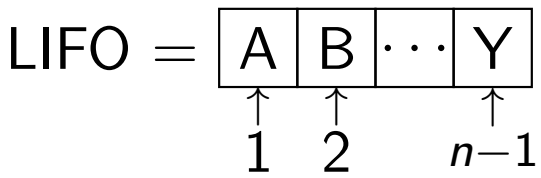
Tipos de listas — FIFO y LIFO

First In First Out



$s \leftarrow$ A

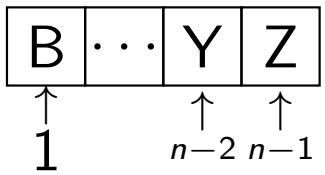
Last In First Out



ADD(FIFO, Y)

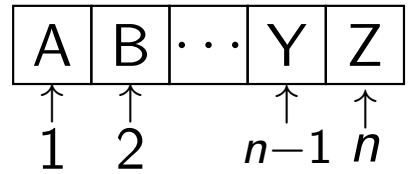
Tipos de listas — FIFO y LIFO

First In First Out

FIFO = 

$s \leftarrow$ A

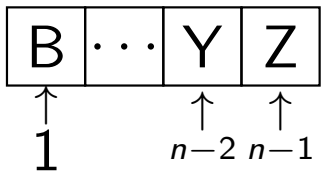
Last In First Out

LIFO = 

ADD(FIFO, Z)

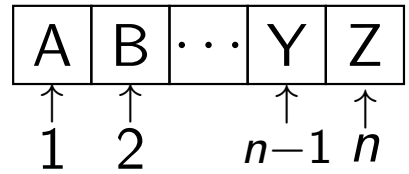
Tipos de listas — FIFO y LIFO

First In First Out

FIFO = 

$s \leftarrow$ A

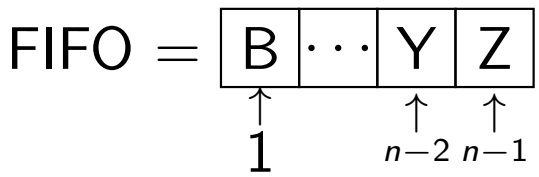
Last In First Out

LIFO = 

$s \leftarrow$ POP(LIFO)

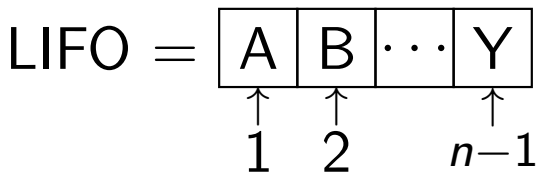
Tipos de listas — FIFO y LIFO

First In First Out



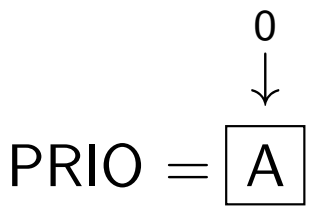
$s \leftarrow$ A

Last In First Out



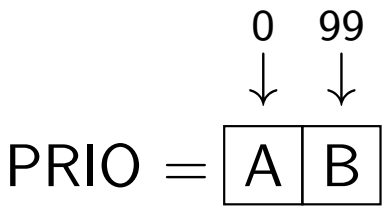
$s \leftarrow$ Z

Tipos de listas — Lista prioritaria



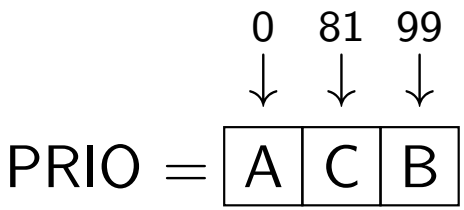
ADD(PRIO, A, 0)

Tipos de listas — Lista prioritaria



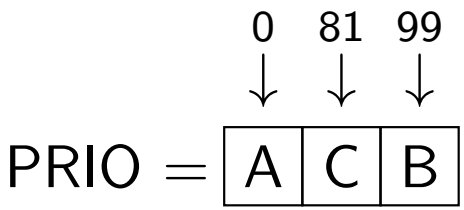
ADD(PRIO, B, 99)

Tipos de listas — Lista prioritaria



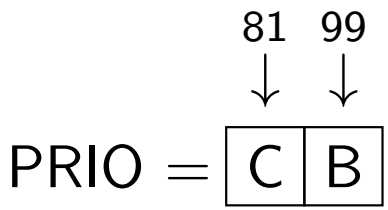
ADD(PRIO, C, 81)

Tipos de listas — Lista prioritaria



$s \leftarrow \text{POP}(\text{PRIO})$

Tipos de listas — Lista prioritaria

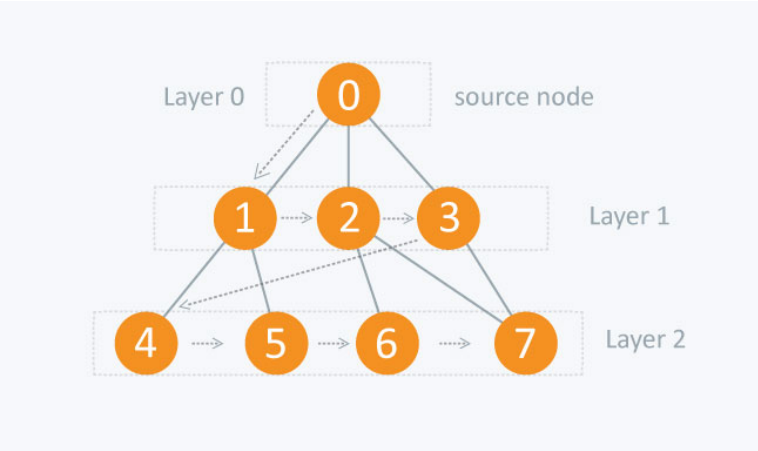


$s \leftarrow$ A

Contenido

MACC
Matemáticas Aplicadas y
Ciencias de la Computación

Algoritmo primero en anchura



Se busca un estado objetivo un nivel a la vez. Se expande primero el nodo de menor altura en la frontera.

Ventajas: Siempre encuentra una solución. Es una solución óptima en términos de número de acciones.

Desventajas: Crecimiento exponencial en el uso de recursos computacionales.

MACC
Matemáticas Aplicadas y Ciencias de la Computación

Algoritmo primero en anchura

Frontera
FIFO =

Algoritmo primero en anchura

Frontera
FIFO = $\boxed{\text{A}}$

Algoritmo primero en anchura

Frontera
FIFO = $\boxed{\text{A}}$

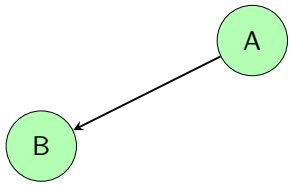
estado $\leftarrow$ POP(Frontera)

Algoritmo primero en anchura

Frontera
FIFO $=$

estado $\leftarrow$ A

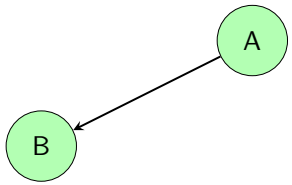
Algoritmo primero en anchura



Frontera
FIFO $=$

estado $\leftarrow$ A
hijo $\leftarrow$ B (test $=$ False)

Algoritmo primero en anchura

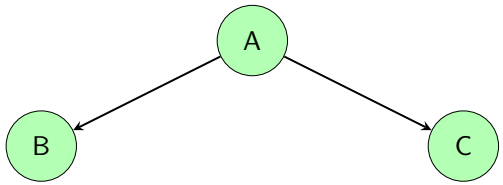


Frontera
FIFO = [B]

estado $\leftarrow$ A

ADD(*frontera*, B)

Algoritmo primero en anchura

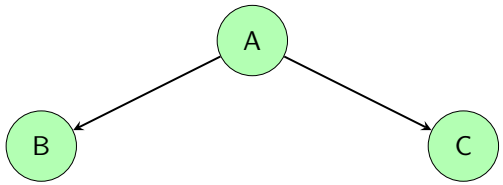


Frontera
FIFO = [B]

estado $\leftarrow$ A

hijo $\leftarrow$ C (test = False)

Algoritmo primero en anchura



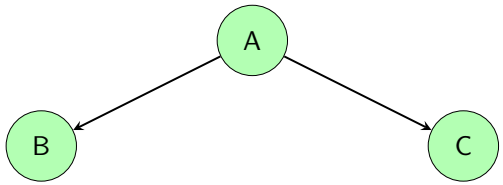
Frontera

FIFO = | B | C |

estado $\leftarrow$ A

ADD(*frontera*, C)

Algoritmo primero en anchura

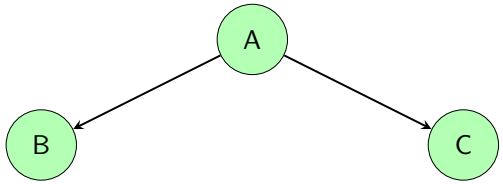


Frontera

FIFO = | B | C |

estado $\leftarrow$ POP(Frontera)

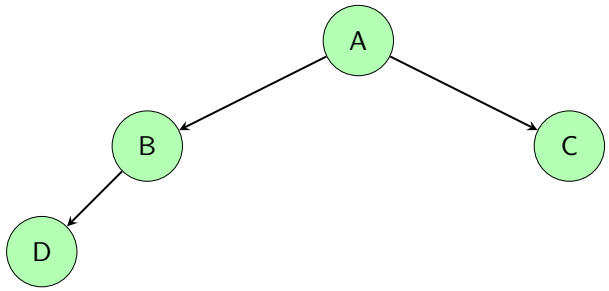
Algoritmo primero en anchura



Frontera
FIFO = [C]

estado ← B

Algoritmo primero en anchura

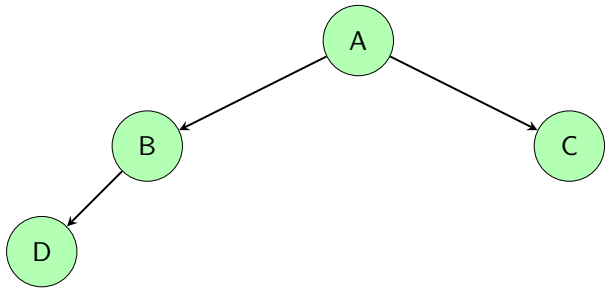


Frontera
FIFO = $\boxed{\text{C}}$

estado $\leftarrow$ B

hijo $\leftarrow$ D (test = False)

Algoritmo primero en anchura



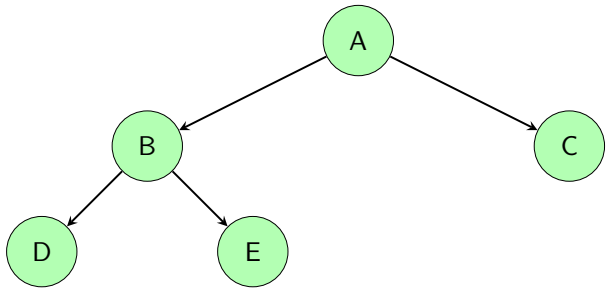
Frontera

FIFO = | C | D |

estado $\leftarrow$ B

ADD(*frontera*, D)

Algoritmo primero en anchura



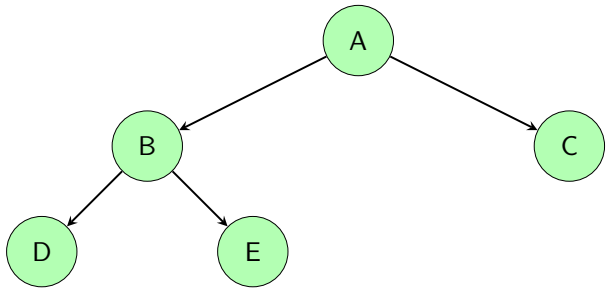
Frontera

FIFO = | C | D |

estado $\leftarrow$ B

hijo $\leftarrow$ E (test = False)

Algoritmo primero en anchura



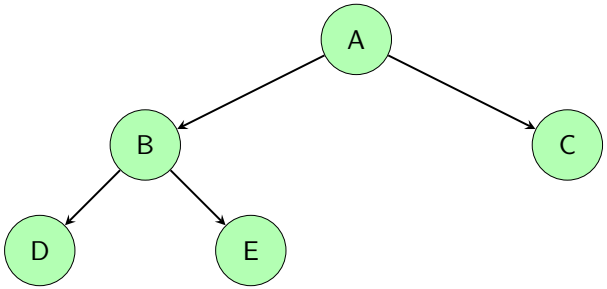
Frontera

FIFO = | C | D | E |

estado $\leftarrow$ B

ADD(*frontera*, E)

Algoritmo primero en anchura

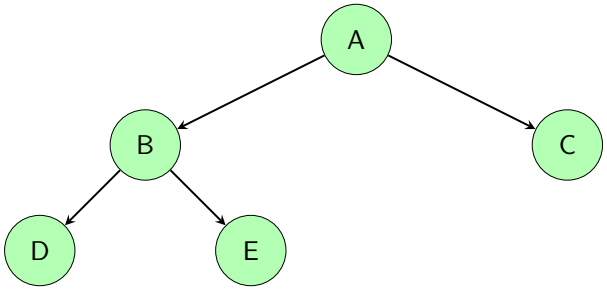


Frontera

FIFO = | C | D | E |

estado $\leftarrow$ POP(Frontera)

Algoritmo primero en anchura

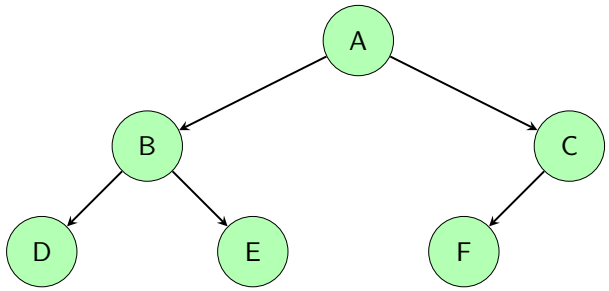


Frontera

FIFO = | D | E |

estado $\leftarrow$ C

Algoritmo primero en anchura



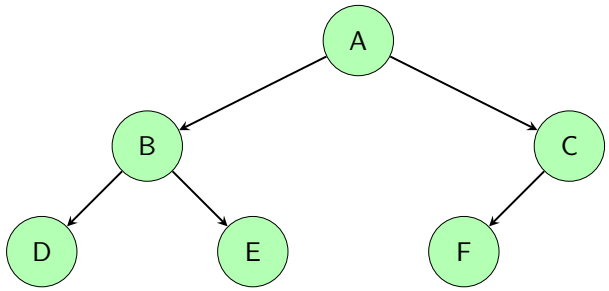
Frontera

FIFO = | D | E |

estado $\leftarrow$ C

hijo $\leftarrow$ F (test = False)

Algoritmo primero en anchura



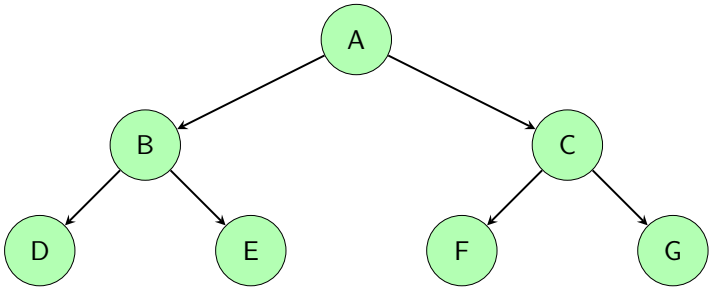
Frontera

FIFO = | D | E | F |

estado $\leftarrow$ C

ADD(*frontera*, F)

Algoritmo primero en anchura



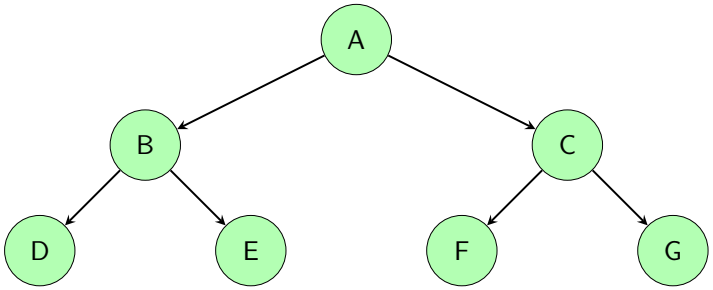
Frontera

FIFO = | D | E | F |

estado $\leftarrow$ C

hijo $\leftarrow$ G (test = False)

Algoritmo primero en anchura



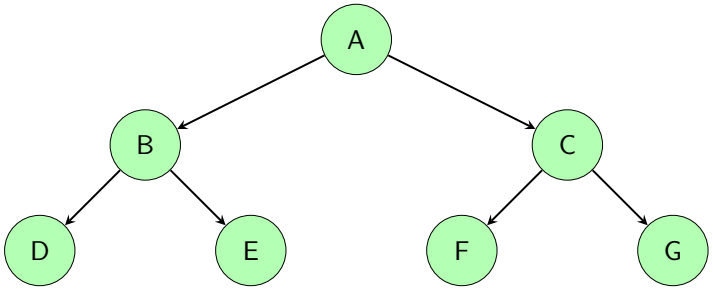
Frontera

FIFO = | D | E | F | G |

estado $\leftarrow$ C

ADD(*frontera*, G)

Algoritmo primero en anchura

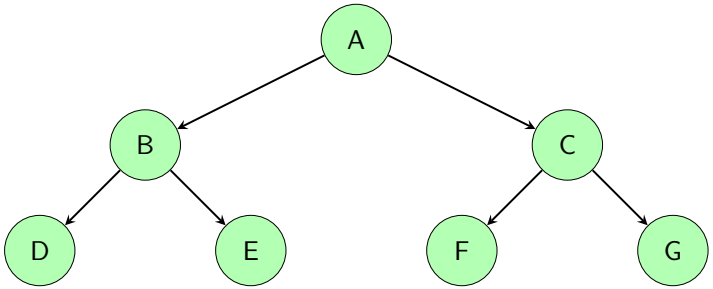


Frontera

FIFO = | D | E | F | G |

estado $\leftarrow$ POP(Frontera)

Algoritmo primero en anchura

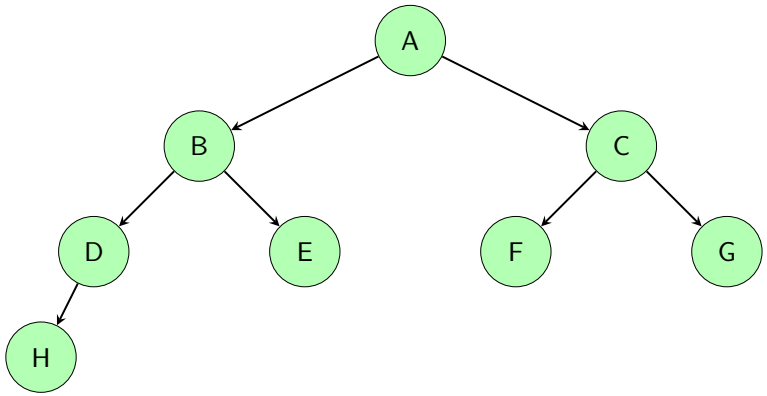


Frontera

FIFO = | E | F | G |

estado $\leftarrow$ D

Algoritmo primero en anchura



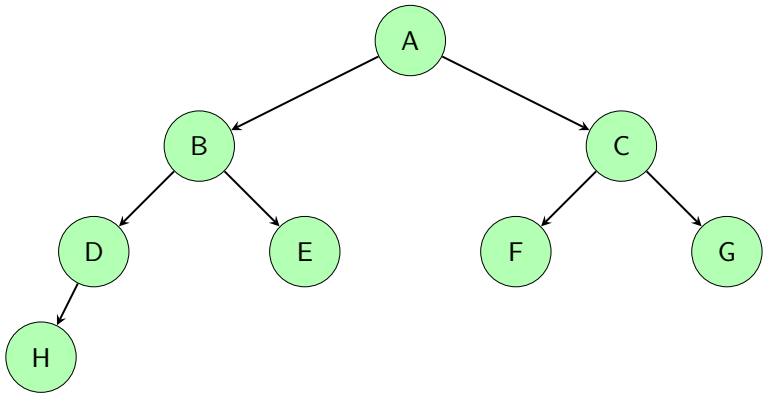
Frontera

FIFO = | E | F | G |

estado $\leftarrow$ D

hijo $\leftarrow$ H (test = False)

Algoritmo primero en anchura



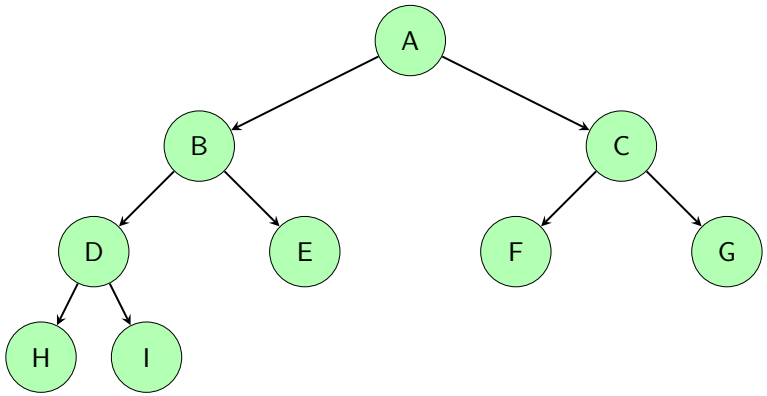
Frontera

FIFO = | E | F | G | H |

estado $\leftarrow$ D

ADD(*frontera*, H)

Algoritmo primero en anchura



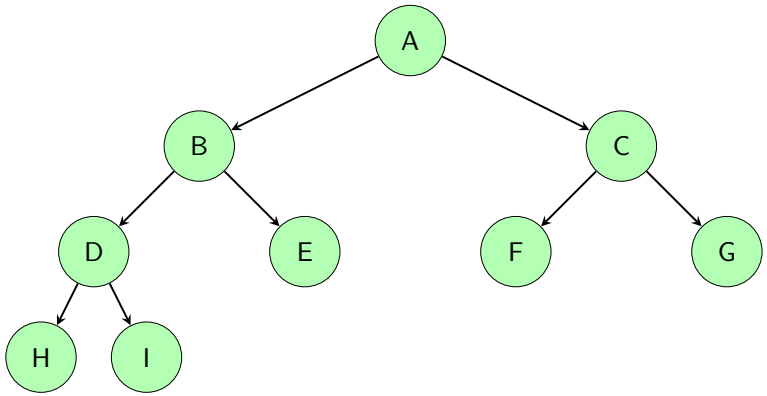
Frontera

FIFO = | E | F | G | H |

estado $\leftarrow$ D

hijo $\leftarrow$ I (test = False)

Algoritmo primero en anchura



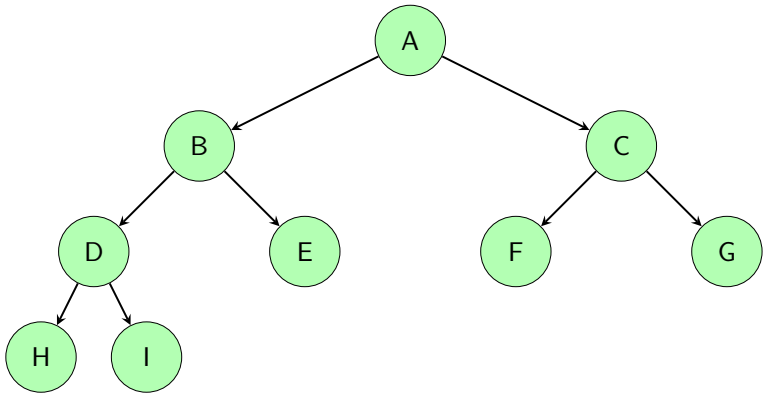
Frontera

FIFO = | E | F | G | H | I |

estado $\leftarrow$ D

ADD(*frontera*, I)

Algoritmo primero en anchura

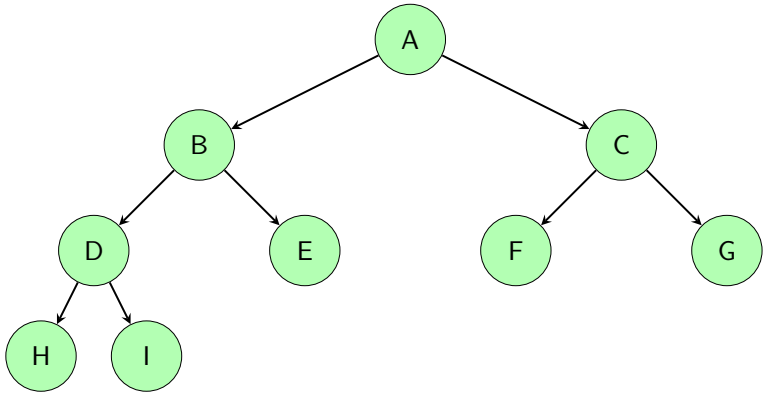


Frontera

FIFO = | E | F | G | H | I |

estado $\leftarrow$ POP(Frontera)

Algoritmo primero en anchura

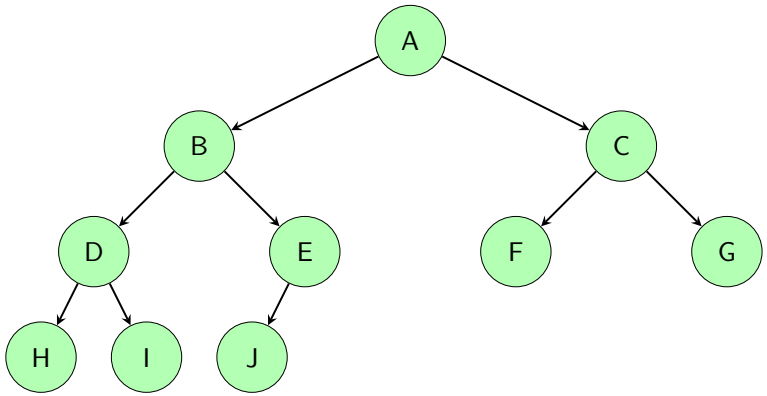


Frontera

FIFO = | F | G | H | I |

estado $\leftarrow$ E

Algoritmo primero en anchura



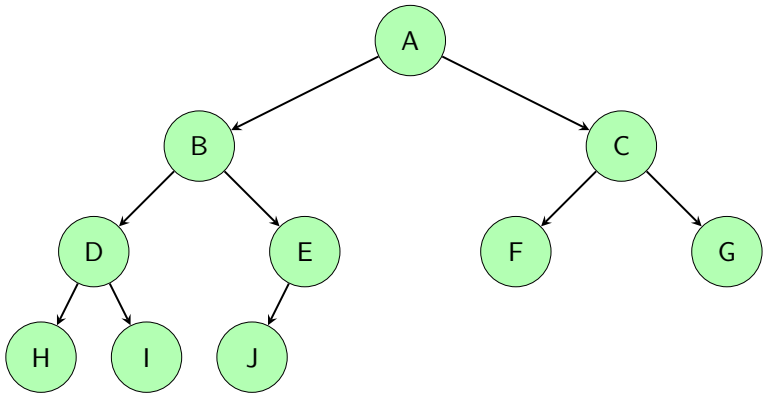
Frontera

FIFO = | F | G | H | I |

estado $\leftarrow$ E

hijo $\leftarrow$ J (test = False)

Algoritmo primero en anchura



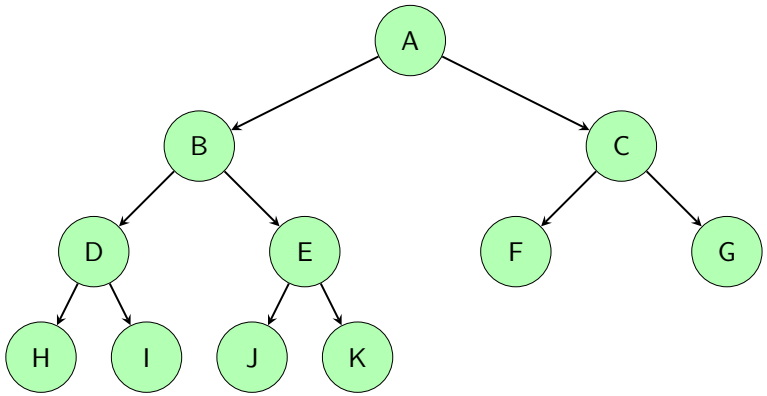
Frontera

FIFO = | F | G | H | I | J |

estado $\leftarrow$ E

ADD(*frontera*, J)

Algoritmo primero en anchura



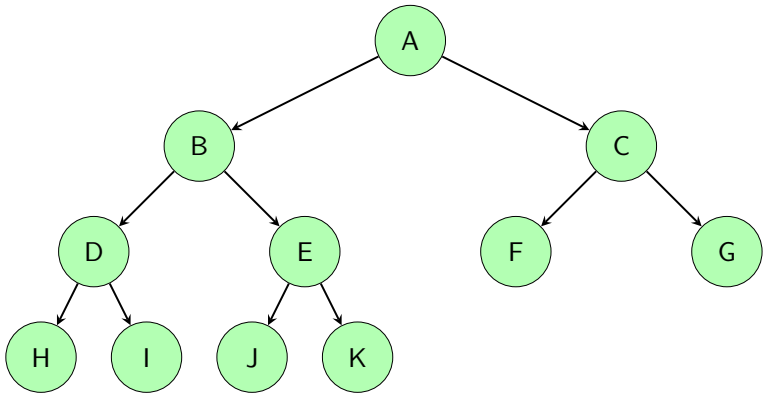
Frontera

FIFO = | F | G | H | I | J |

estado $\leftarrow$ E

hijo $\leftarrow$ K (test = False)

Algoritmo primero en anchura



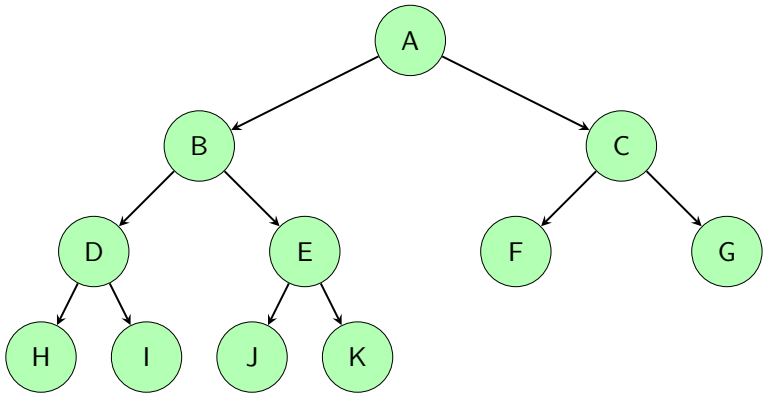
Frontera

FIFO = | F | G | H | I | J | K |

estado ← E

ADD(*frontera*, K)

Algoritmo primero en anchura

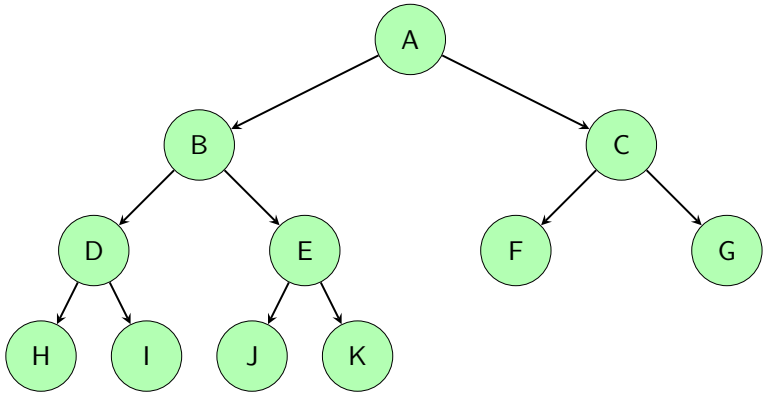


Frontera

FIFO = | F | G | H | I | J | K |

estado $\leftarrow$ POP(Frontera)

Algoritmo primero en anchura

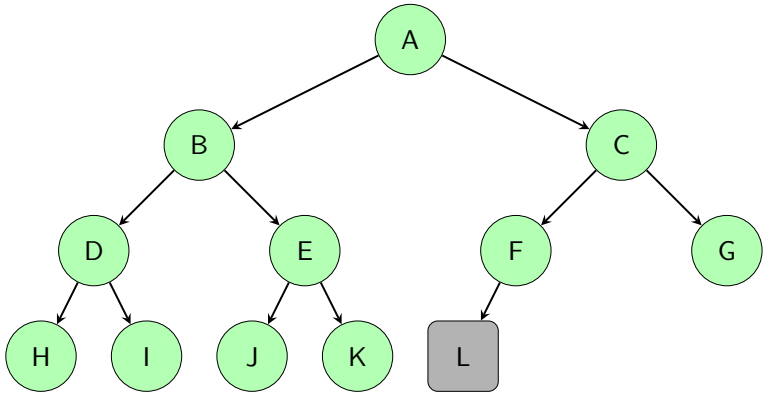


Frontera

FIFO = | G | H | I | J | K |

estado $\leftarrow$ F

Algoritmo primero en anchura



Frontera

FIFO = | G | H | I | J | K |

estado $\leftarrow$ F

hijo $\leftarrow$ L (test = True)

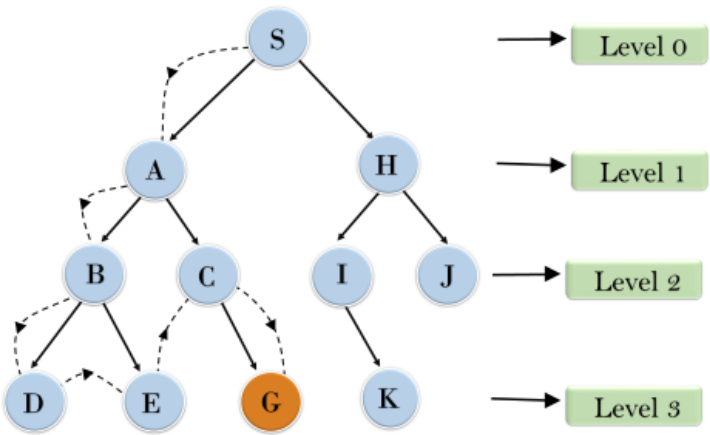
Complejidad computacional

Depth	Nodes	Time	Memory
2	110	.11 milliseconds	107 kilobytes
4	11,110	11 milliseconds	10.6 megabytes
6	10^6	1.1 seconds	1 gigabyte
8	10^8	2 minutes	103 gigabytes
10	10^{10}	3 hours	10 terabytes
12	10^{12}	13 days	1 petabyte
14	10^{14}	3.5 years	99 petabytes
16	10^{16}	350 years	10 exabytes

Figure 3.13 Time and memory requirements for breadth-first search. The numbers shown assume branching factor $b = 10$; 1 million nodes/second; 1000 bytes/node.

MACC
Matemáticas Aplicadas y Ciencias de la Computación

Algortimo primero en profundidad



Se busca un estado objetivo siguiendo un camino hacia abajo en niveles. Se expande primero el nodo de mayor altura en la frontera.

Ventajas: Si se tiene suerte, se debe explorar solo una parte pequeña del espacio.

Desventajas: La solución encontrada no necesariamente es óptima.

Algoritmo primero en profundidad

Frontera
LIFO =

Algoritmo primero en profundidad

Frontera
LIFO = [A]

Algoritmo primero en profundidad

Frontera
LIFO = $\boxed{\text{A}}$

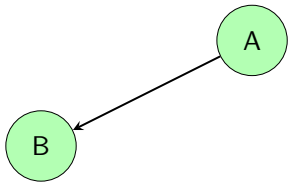
estado $\leftarrow$ POP(Frontera)

Algoritmo primero en profundidad

Frontera
LIFO $=$

estado $\leftarrow$ A

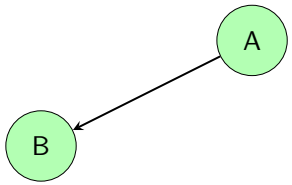
Algoritmo primero en profundidad



Frontera
LIFO =

estado $\leftarrow$ A
hijo $\leftarrow$ B (test = False)

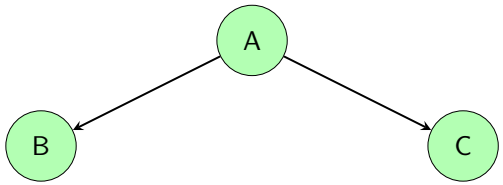
Algoritmo primero en profundidad



Frontera
LIFO = $\boxed{\text{B}}$

estado $\leftarrow$ A
ADD(*frontera*, B)

Algoritmo primero en profundidad

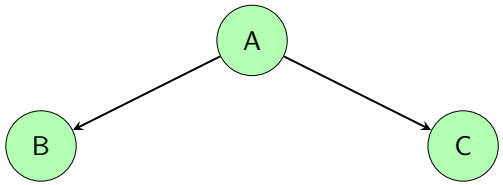


Frontera
LIFO = $\boxed{\text{B}}$

estado $\leftarrow$ A

hijo $\leftarrow$ C (test = False)

Algoritmo primero en profundidad



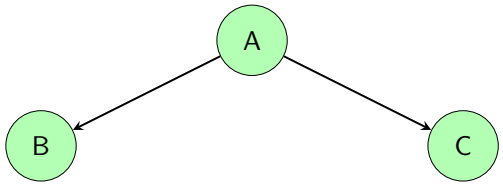
Frontera

LIFO = | B | C |

estado $\leftarrow$ A

ADD(*frontera*, C)

Algoritmo primero en profundidad

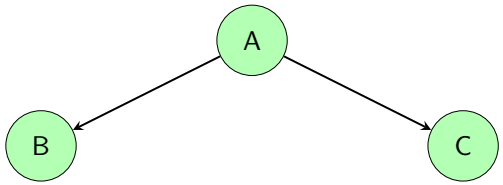


Frontera

LIFO = | B | C |

estado $\leftarrow$ POP(Frontera)

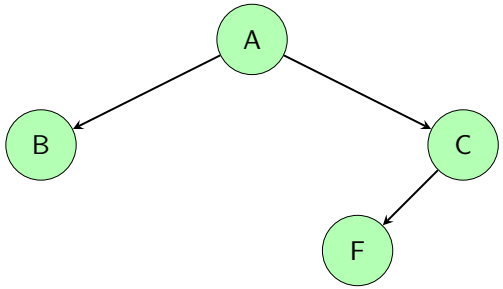
Algoritmo primero en profundidad



Frontera
LIFO = [B]

estado ← C

Algoritmo primero en profundidad



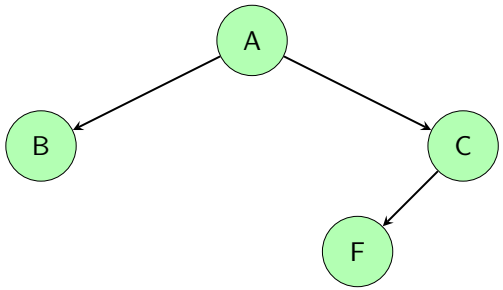
Frontera

LIFO = $\boxed{\text{B}}$

estado $\leftarrow$ C

hijo $\leftarrow$ F (test = False)

Algoritmo primero en profundidad



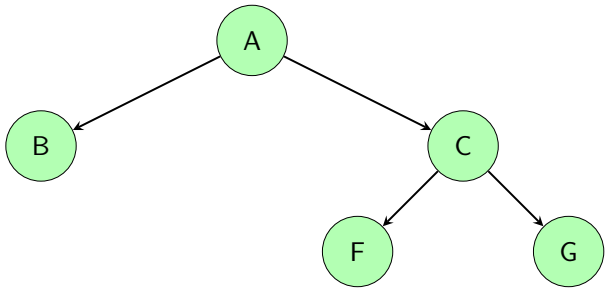
Frontera

LIFO = | B | F |

estado $\leftarrow$ C

ADD(*frontera*, F)

Algoritmo primero en profundidad



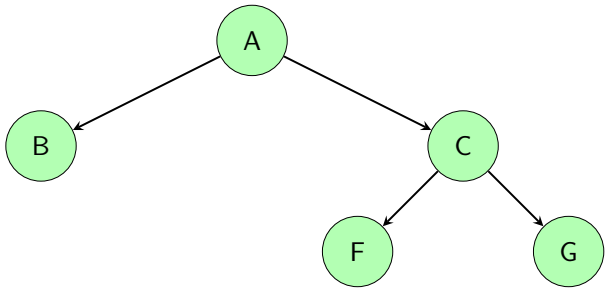
Frontera

LIFO = | B | F |

estado $\leftarrow$ C

hijo $\leftarrow$ G (test = False)

Algoritmo primero en profundidad



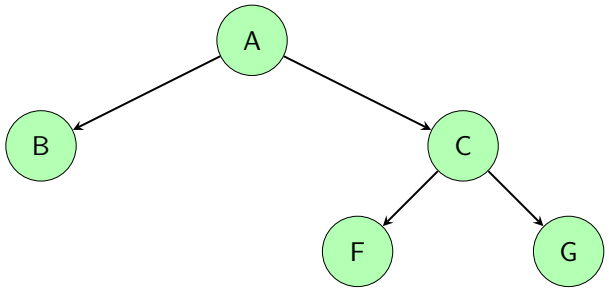
Frontera

LIFO = | B | F | G |

estado $\leftarrow$ C

ADD(*frontera*, G)

Algoritmo primero en profundidad

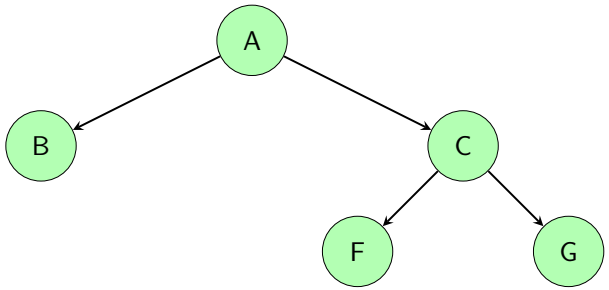


Frontera

LIFO = | B | F | G |

estado $\leftarrow$ POP(Frontera)

Algoritmo primero en profundidad

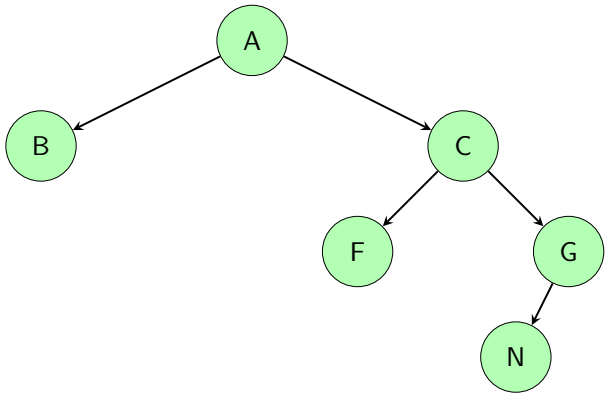


Frontera

LIFO = | B | F |

estado ← G

Algoritmo primero en profundidad



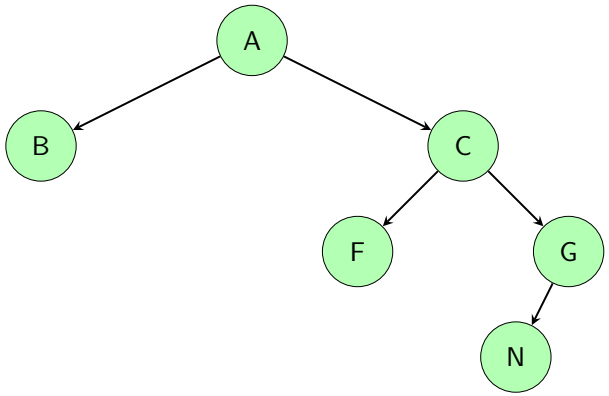
Frontera

LIFO = | B | F |

estado $\leftarrow$ G

hijo $\leftarrow$ N (test = False)

Algoritmo primero en profundidad



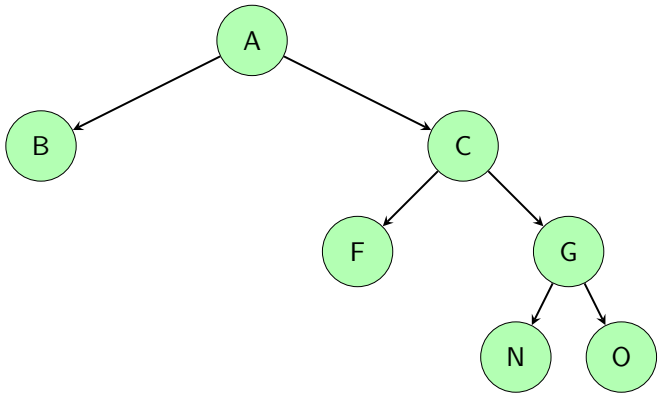
Frontera

LIFO = | B | F | N |

estado $\leftarrow$ G

ADD(*frontera*, N)

Algoritmo primero en profundidad



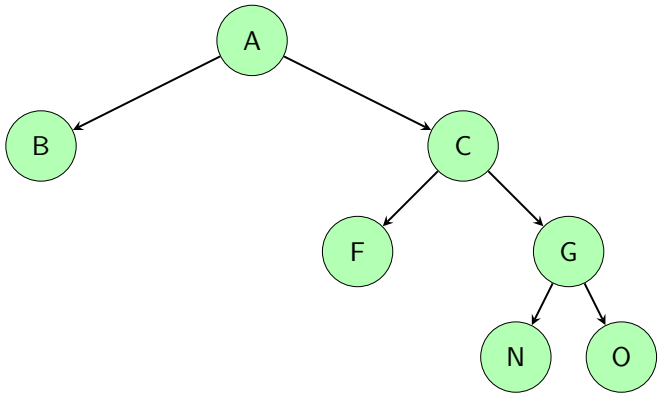
Frontera

LIFO = | B | F | N |

estado $\leftarrow$ G

hijo $\leftarrow$ O (test = False)

Algoritmo primero en profundidad



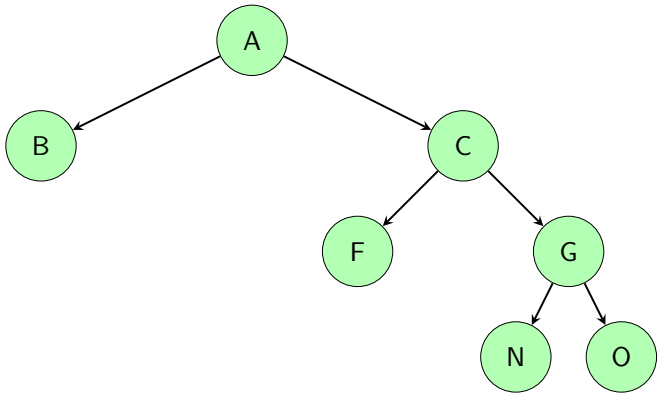
Frontera

LIFO = | B | F | N | O |

estado $\leftarrow$ G

ADD(*frontera*, O)

Algoritmo primero en profundidad

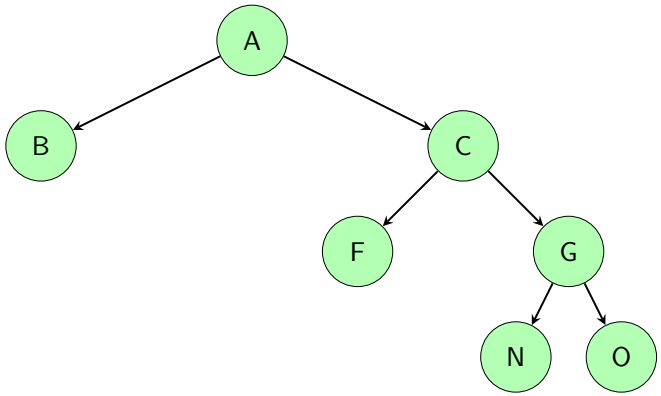


Frontera

LIFO = | B | F | N | O |

estado $\leftarrow$ POP(Frontera)

Algoritmo primero en profundidad

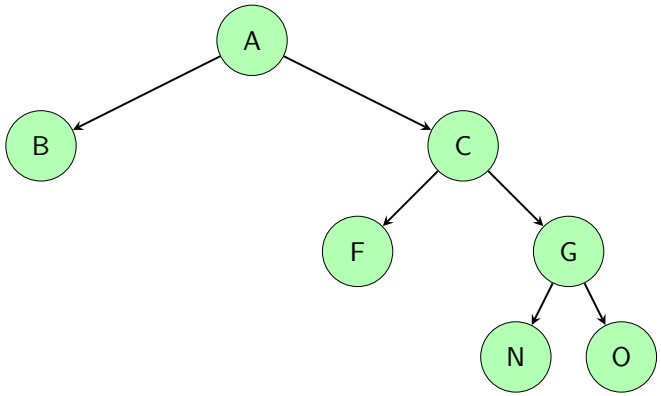


Frontera

LIFO = | B | F | N |

estado $\leftarrow$ O

Algoritmo primero en profundidad

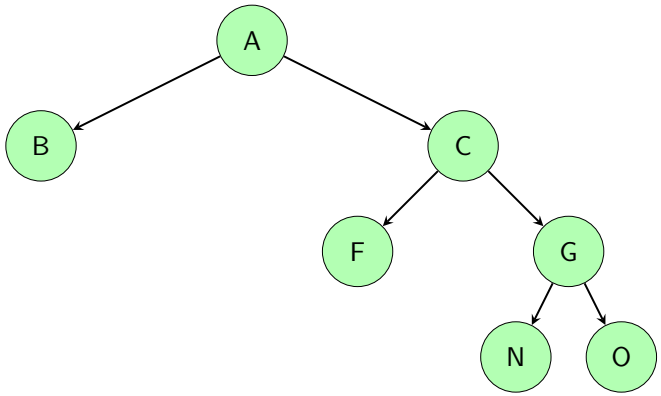


Frontera

LIFO = | B | F | N |

estado $\leftarrow$ POP(Frontera)

Algoritmo primero en profundidad

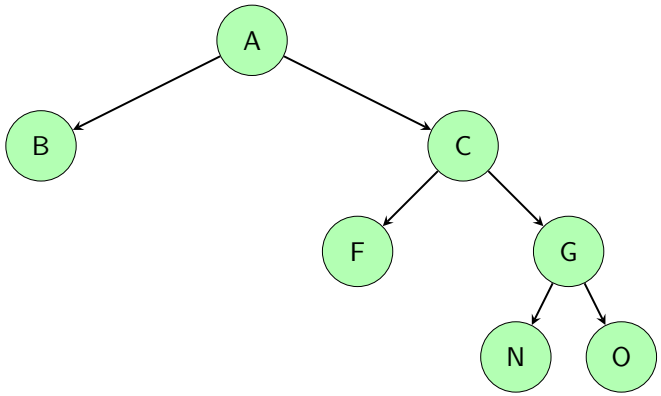


Frontera

LIFO = | B | F |

estado $\leftarrow$ N

Algoritmo primero en profundidad

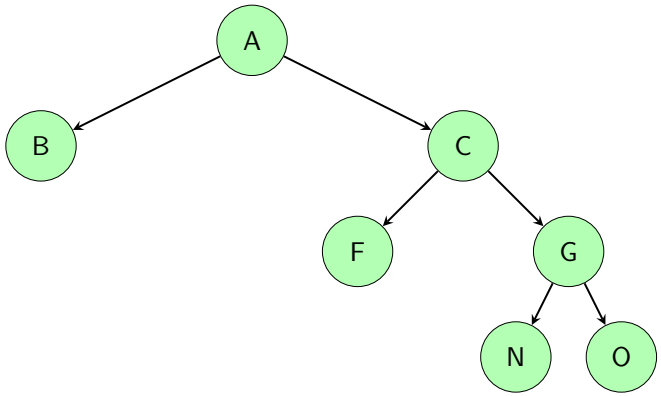


Frontera

LIFO = | B | F |

estado $\leftarrow$ POP(Frontera)

Algoritmo primero en profundidad

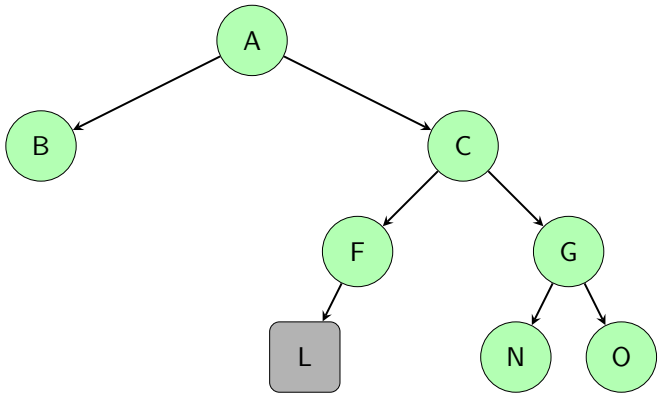


Frontera

LIFO = $\boxed{\text{B}}$

estado $\leftarrow$ F

Algoritmo primero en profundidad



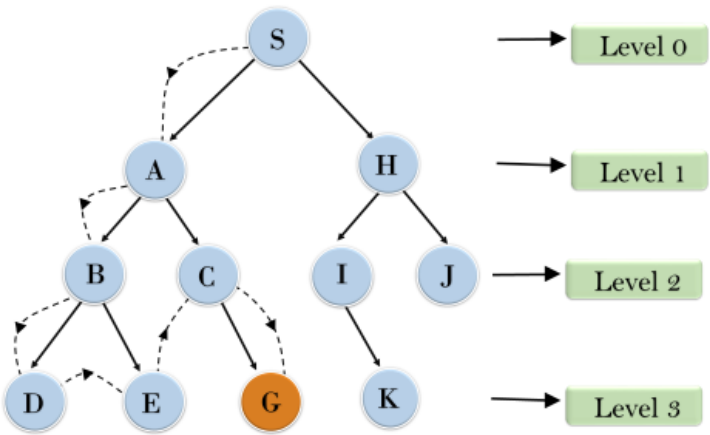
Frontera

LIFO = $\boxed{\text{B}}$

estado $\leftarrow$ F

hijo $\leftarrow$ L (test = True)

Algortimo backtracking



Se busca un estado objetivo siguiendo un camino hacia abajo en niveles mediante una llamada recursiva.

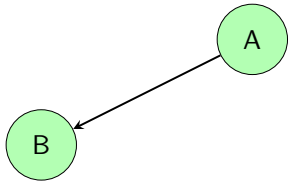
Ventajas: Usa recursos computacionales de complejidad lineal (solo se guarda el camino actual). Si se tiene suerte, se debe explorar solo una parte pequeña del espacio.

Desventajas: La solución encontrada no necesariamente es óptima.

Algoritmo backtracking

backtracking(A) = ?

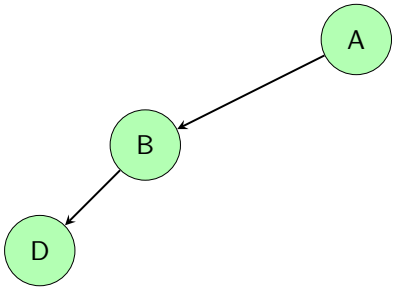
Algoritmo backtracking



backtracking(A) = ?

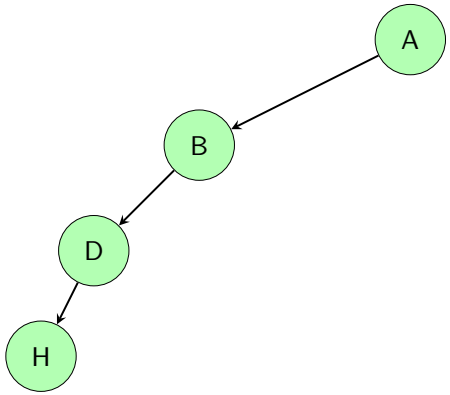
backtracking(B) = ?

Algoritmo backtracking



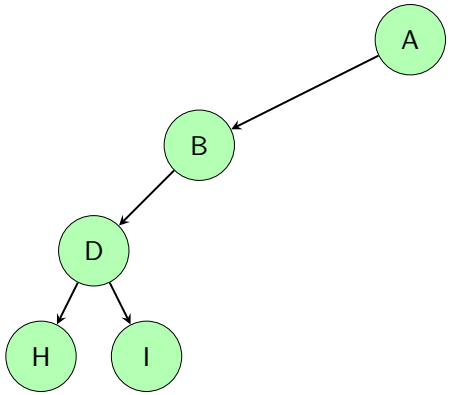
$$\text{backtracking}(A) = ?$$
$$\text{backtracking}(B) = ?$$
$$\text{backtracking}(D) = ?$$

Algoritmo backtracking



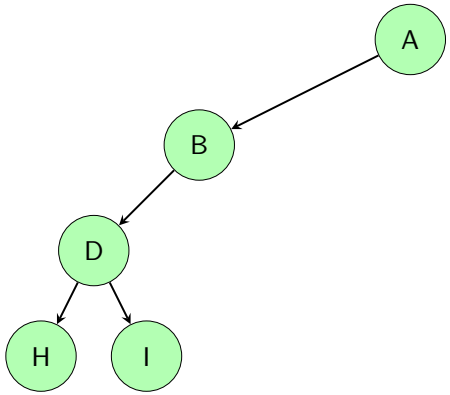
$$\text{backtracking}(A) = ?$$
$$\text{backtracking}(B) = ?$$
$$\text{backtracking}(D) = ?$$
$$\text{backtracking}(H) = \text{falla}$$

Algoritmo backtracking



backtracking(A) = ?
backtracking(B) = ?
backtracking(D) = ?
backtracking(H) = falla
backtracking(I) = falla

Algoritmo backtracking

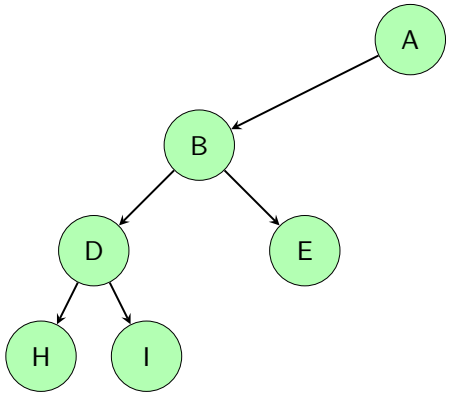


backtracking(A) = ?

backtracking(B) = ?

backtracking(D) = falla

Algoritmo backtracking



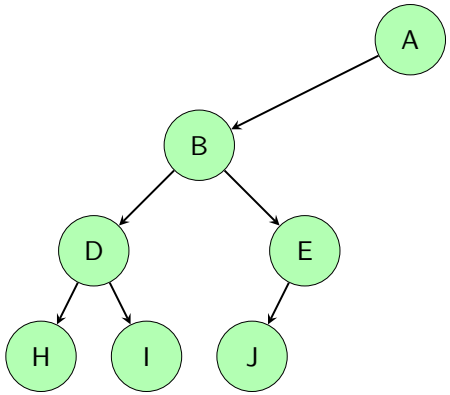
backtracking(A) = ?

backtracking(B) = ?

backtracking(D) = falla

backtracking(E) = ?

Algoritmo backtracking



backtracking(A) = ?

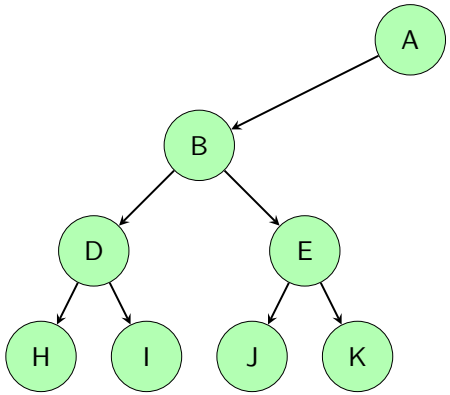
backtracking(B) = ?

backtracking(D) = falla

backtracking(E) = ?

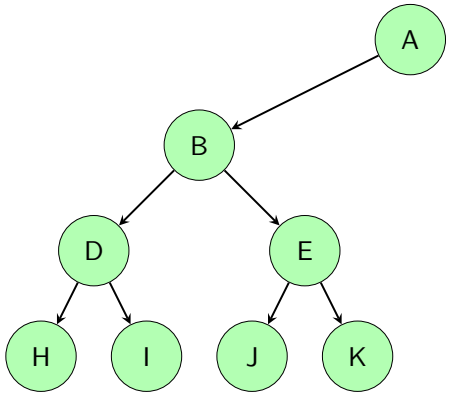
backtracking(J) = falla

Algoritmo backtracking



backtracking(A) = ?
    backtracking(B) = ?
        backtracking(D) = falla
        backtracking(E) = ?
            backtracking(J) = falla
            backtracking(K) = falla

Algoritmo backtracking



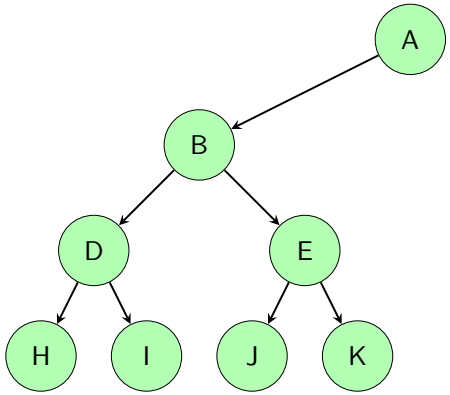
backtracking(A) = ?

backtracking(B) = ?

backtracking(D) = falla

backtracking(E) = falla

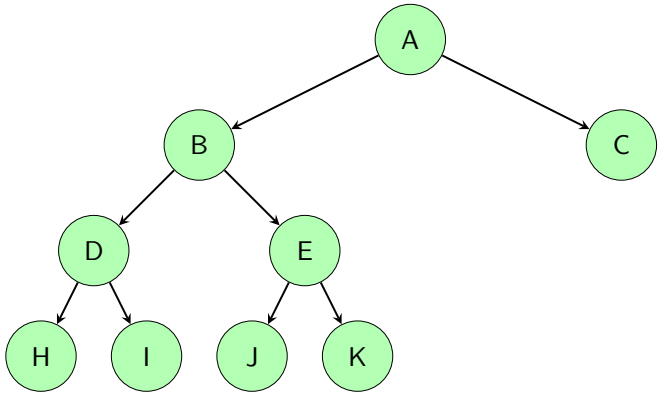
Algoritmo backtracking



backtracking(A) = ?

backtracking(B) = falla

Algoritmo backtracking

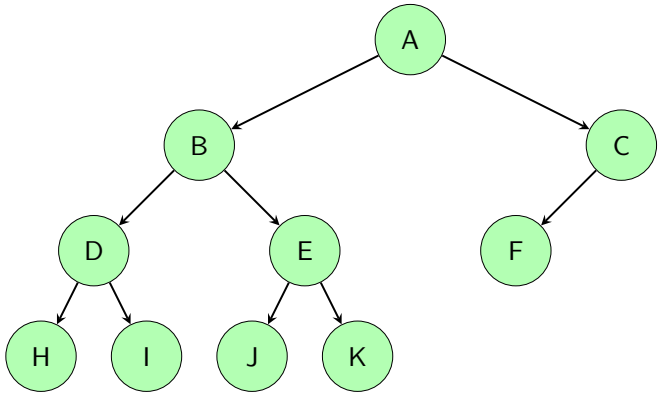


backtracking(A) = ?

backtracking(B) = falla

backtracking(C) = ?

Algoritmo backtracking



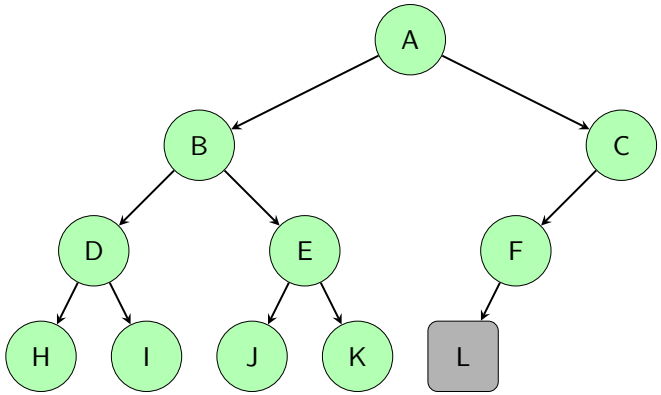
backtracking(A) = ?

backtracking(B) = falla

backtracking(C) = ?

backtracking(F) = ?

Algoritmo backtracking



backtracking(A) = ?

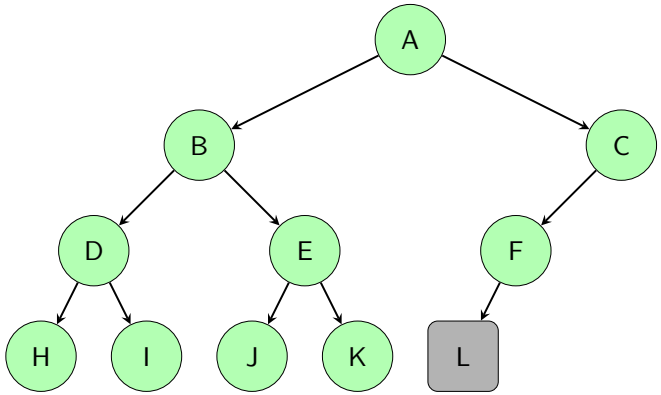
backtracking(B) = falla

backtracking(C) = ?

backtracking(F) = ?

backtracking(L) = L

Algoritmo backtracking



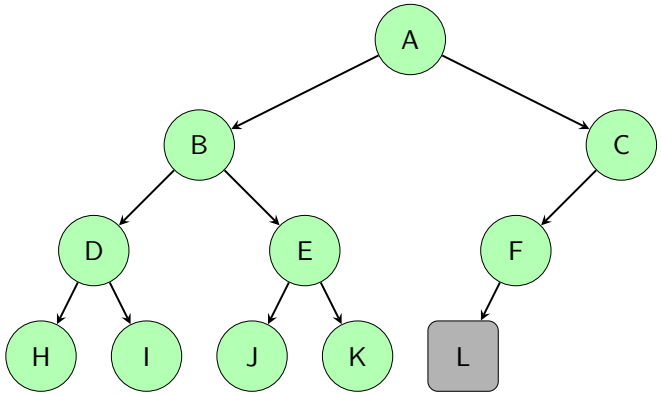
backtracking(A) = ?

backtracking(B) = falla

backtracking(C) = ?

backtracking(F) = L

Algoritmo backtracking

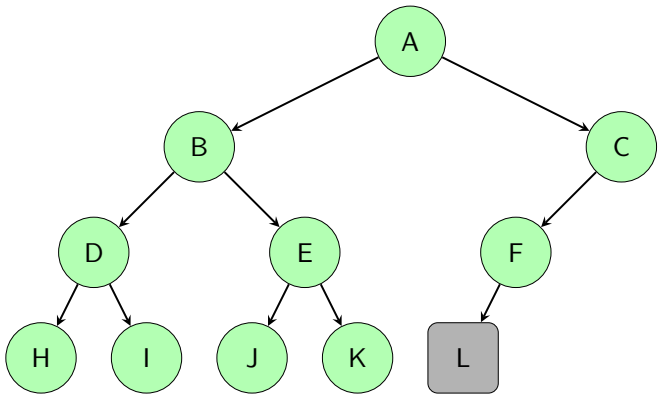


backtracking(A) = ?

backtracking(B) = falla

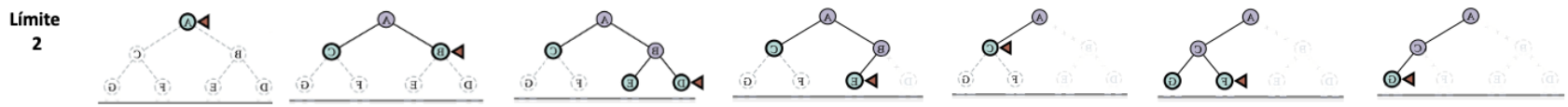
backtracking(C) = L

Algoritmo backtracking



backtracking(A) = L

Algortimo profundidad limitada



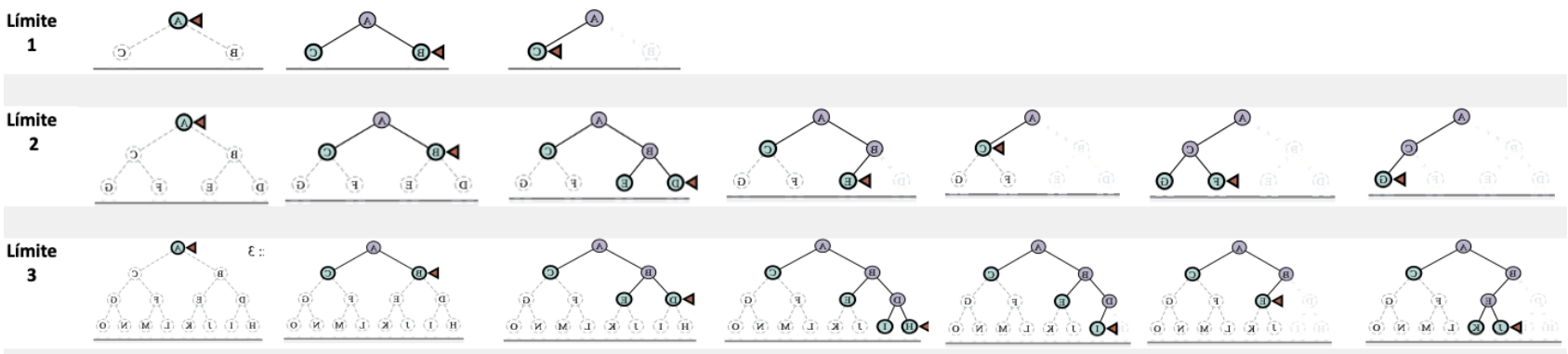
Se establece un límite a la profundidad de los nodos. Se expande primero el nodo de mayor altura en la frontera. Cuando se explora el primer nodo que supera el límite de profundidad, se corta la exploración por esa rama (como si el nodo no tuviera acciones aplicables).

Ventajas: Usa menos recursos que backtracking.

Desventajas: No siempre encuentra una solución.

MACC
Matemáticas Aplicadas y Ciencias de la Computación

Algortimo profundización iterativa



Se realiza profundidad limitada cada vez aumentando el límite de profundidad.

Ventajas: Uso de memoria es de complejidad lineal (solo se guarda el camino actual). Si encuentra una solución, es óptima.

Desventajas: Uso de tiempo es exponencial.

MACC
Matemáticas Aplicadas y Ciencias de la Computación

Contenido

MACC
Matemáticas Aplicadas y Ciencias de la Computación

Eficiencia de un algoritmo

Aspecto teórico:

Complejidad de tiempo: Cantidad de pasos que toma un algoritmo para resolver un problema.

Eficiencia de un algoritmo

Aspecto teórico:

Complejidad de tiempo: Cantidad de pasos que toma un algoritmo para resolver un problema.

Complejidad de espacio: Cantidad de memoria usada por un algoritmo para resolver un problema.

Análisis empírico

☞ Comparamos la eficiencia de dos algoritmos con base en sus tiempos en CPU.

Análisis empírico

☞ Comparamos la eficiencia de dos algoritmos con base en sus tiempos en CPU.

Inconvenientes del proceso:

- La CPU realiza múltiples procesos que pueden afectar la velocidad de ejecución de un algoritmo.

Análisis empírico

☞ Comparamos la eficiencia de dos algoritmos con base en sus tiempos en CPU.

Inconvenientes del proceso:

- La CPU realiza múltiples procesos que pueden afectar la velocidad de ejecución de un algoritmo.
- La velocidad de un algoritmo depende del lenguaje de programación y la compilación interna para su ejecución en CPU.

Análisis empírico

☞ Comparamos la eficiencia de dos algoritmos con base en sus tiempos en CPU.

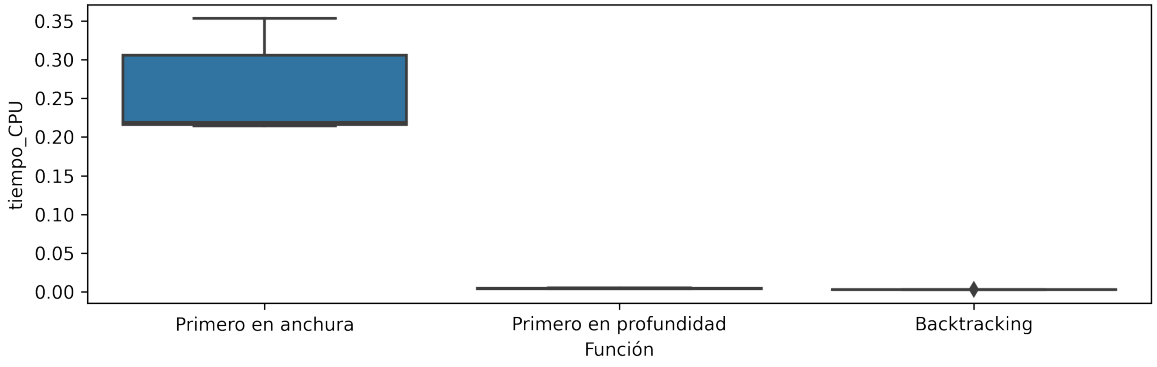
Inconvenientes del proceso:

- La CPU realiza múltiples procesos que pueden afectar la velocidad de ejecución de un algoritmo.
- La velocidad de un algoritmo depende del lenguaje de programación y la compilación interna para su ejecución en CPU.
- P.ej., No todas las implementaciones de ciclos son igualmente rápidas en todos los lenguajes de programación, especialmente en Python.

MACC
Matemáticas Aplicadas y Ciencias de la Computación

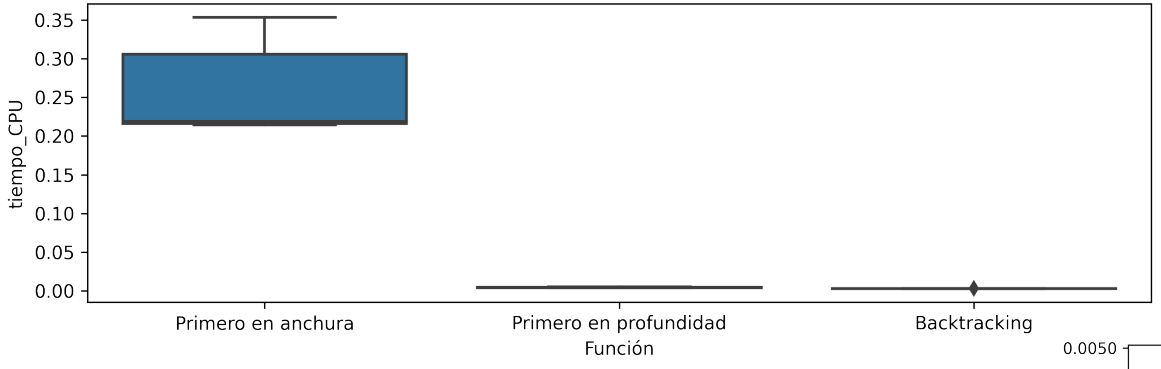
Múltiples mediciones

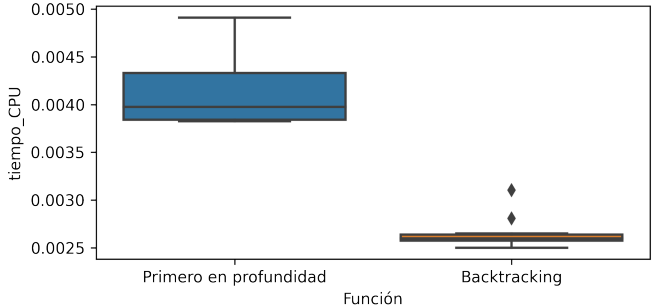
Usamos múltiples mediciones para aproximar la velocidad de ejecución de un algoritmo.



Múltiples mediciones

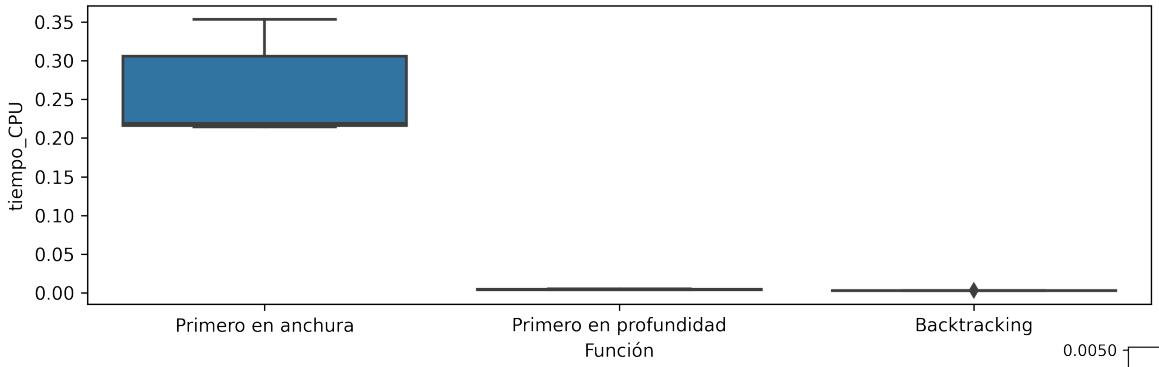
Usamos múltiples mediciones para aproximar la velocidad de ejecución de un algoritmo.



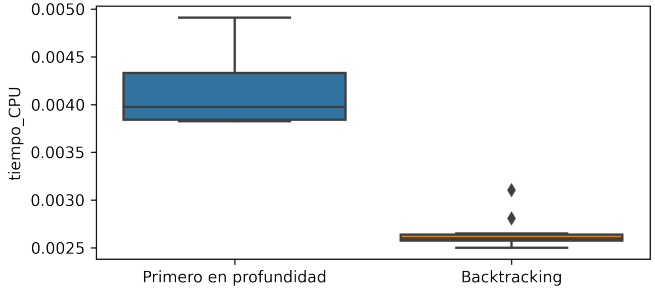


Múltiples mediciones

Usamos múltiples mediciones para aproximar la velocidad de ejecución de un algoritmo.



	tiempo_CPU	
	mean	std
Función		
Backtracking	0.002655	0.000177
Primero en anchura	0.260836	0.059783
Primero en profundidad	0.004162	0.000432



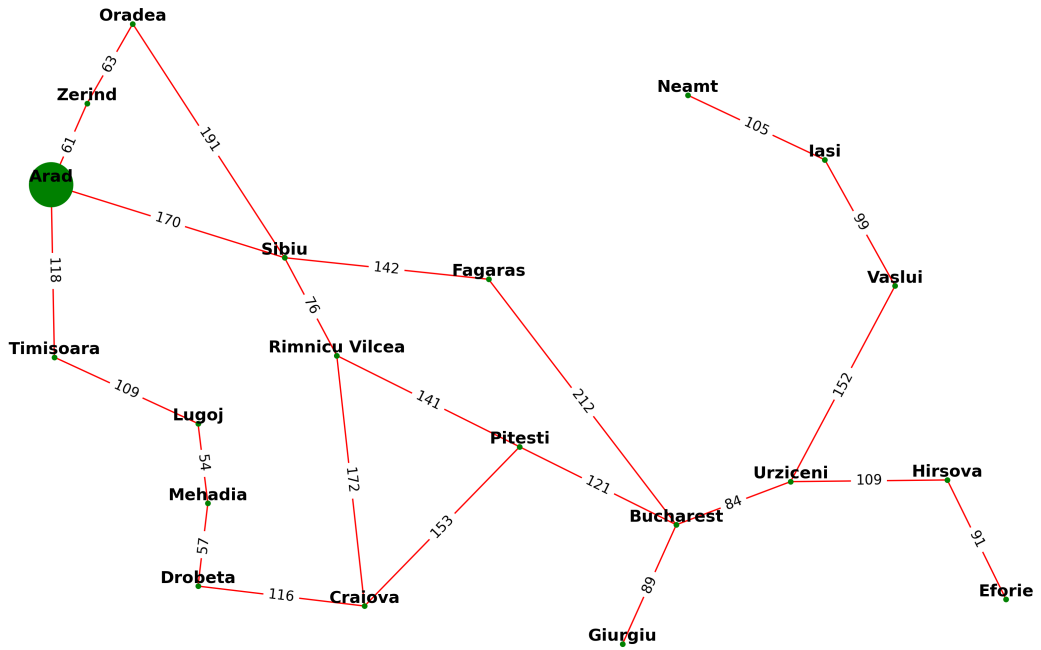
MACC
Matemáticas Aplicadas y Ciencias de la Computación

Contenido

MACC
Matemáticas Aplicadas y Ciencias de la Computación

Comparación algoritmos ciegos

Problema: Ir de Arad a Bucharest.



Comparación algoritmos ciegos

Problema: Ir de Arad a Bucharest.

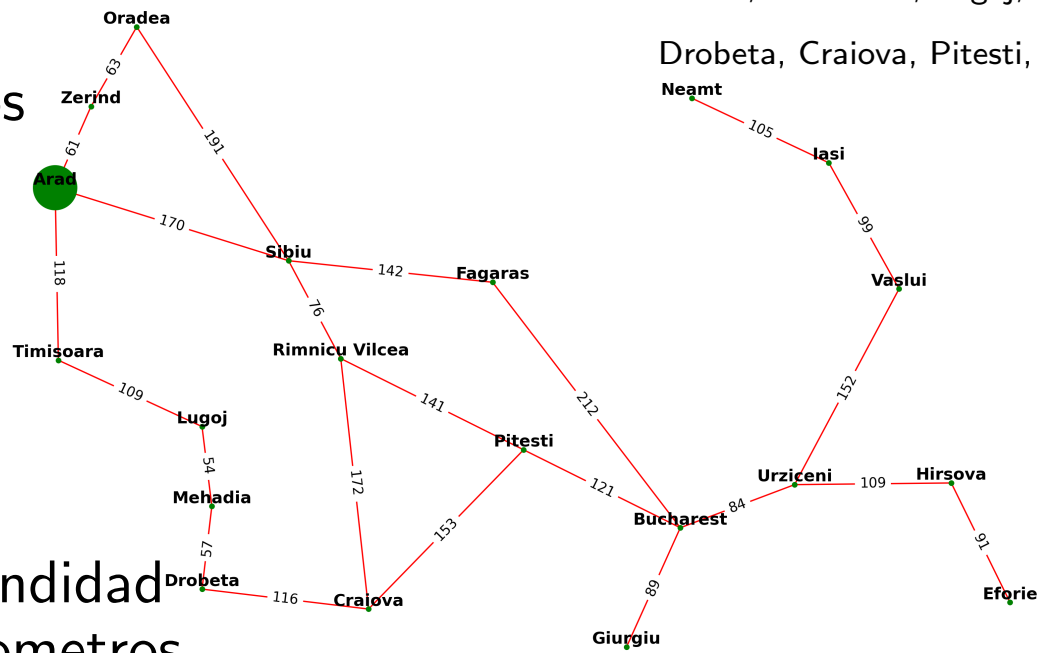
Backtracking
Costo: 728 kilometros

Arad, Timisoara, Lugoj, Mehadia,

Drobeta, Craiova, Pitesti, Bucharest

Primero anchura
Costo: 524 kilometros

Arad, Sibiu, Fagaras, Bucharest

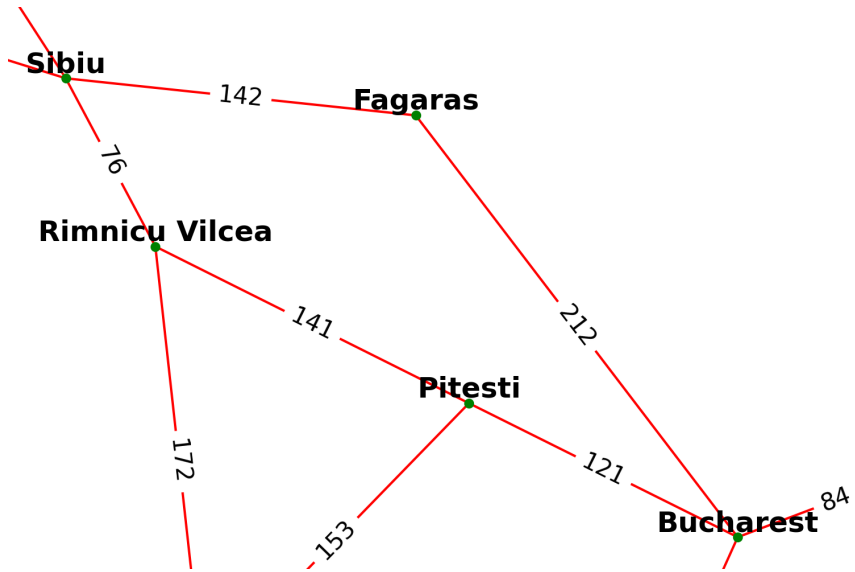


Primero profundidad
Costo: 653 kilometros

Arad, Zerind, Oradea, Sibiu, Rimnicu Vilcea, Pitesti, Bucharest

Matemáticas Aplicadas y Ciencias de la Computación

Pedazo complicado para hallar la ruta óptima



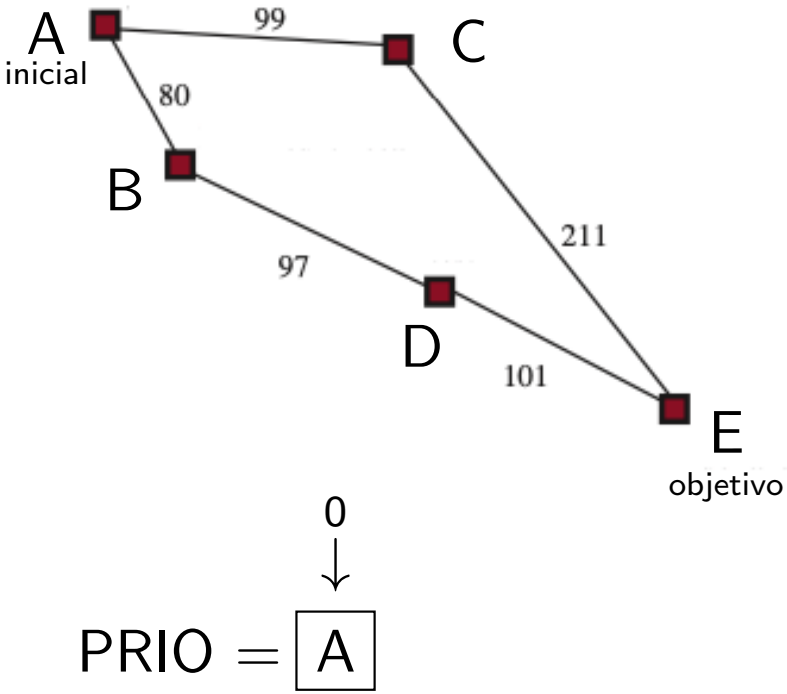
En el trayecto de Sibiu a Bucharest, aunque haya más nodos, es óptimo tomar el camino Rimnicu Vilcea, Pitesti, Bucharest.

MACC
Matemáticas Aplicadas y Ciencias de la Computación

Contenido

MACC
Matemáticas Aplicadas y Ciencias de la Computación

Algoritmo best first search

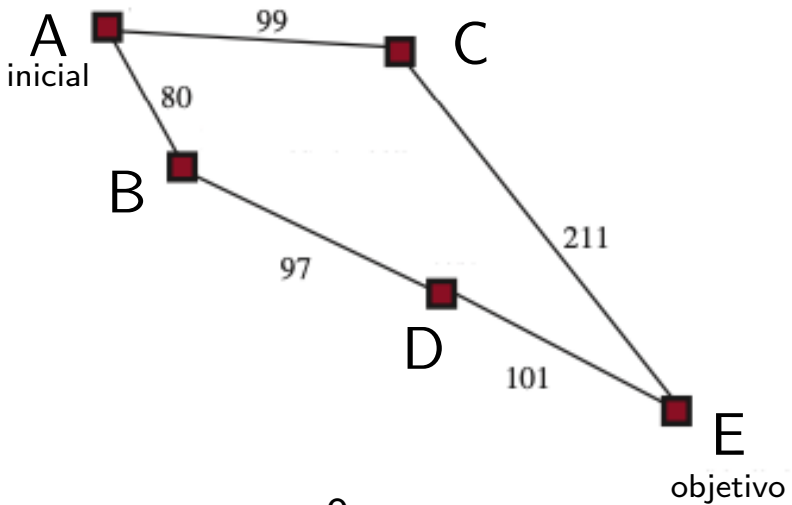


Se busca un estado objetivo siguiendo un camino hacia abajo en niveles. Se expande primero el nodo prioritario en la frontera.

Ventajas: Si se tiene suerte, se debe explorar solo una parte pequeña del espacio. Siempre encuentra la solución óptima.

Desventajas: Uso exponencial de recursos computacionales en el peor caso.

Algoritmo best first search



0

$$PRIO = \boxed{A}$$

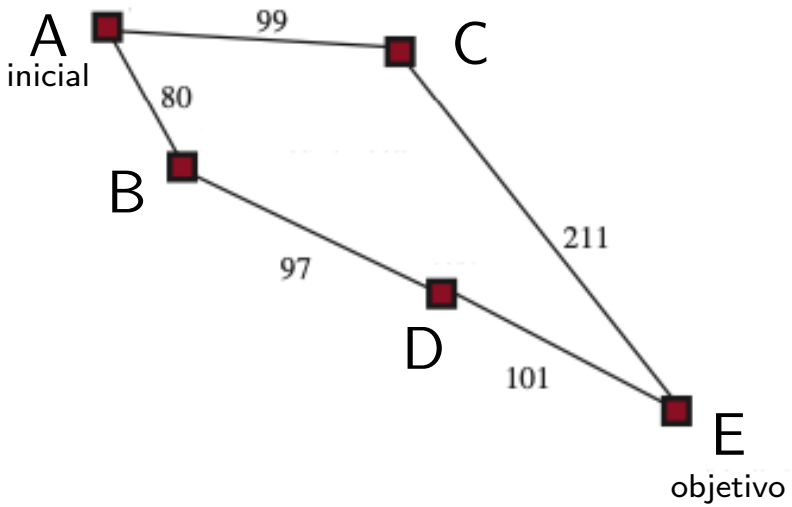
$$nodo \leftarrow \text{POP}(\text{Frontera})$$

Se busca un estado objetivo siguiendo un camino hacia abajo en niveles. Se expande primero el nodo prioritario en la frontera.

Ventajas: Si se tiene suerte, se debe explorar solo una parte pequeña del espacio. Siempre encuentra la solución óptima.

Desventajas: Uso exponencial de recursos computacionales en el peor caso.

Algoritmo best first search



$$\text{PRIO} =$$

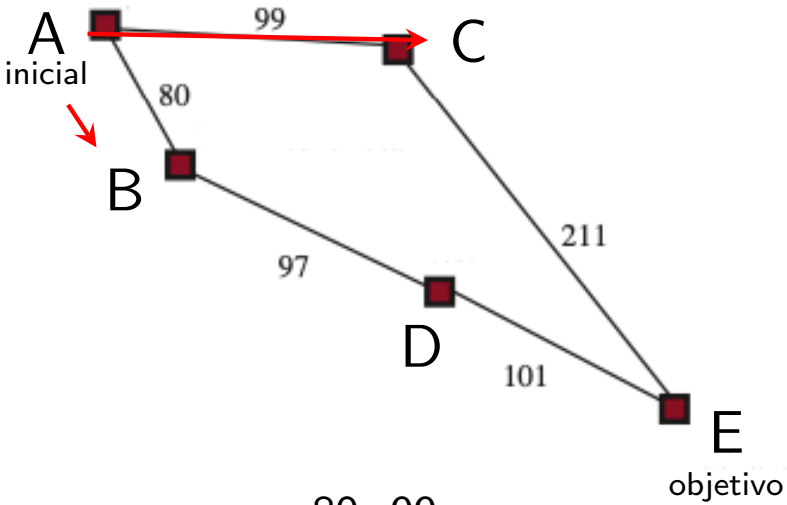
$$nodo \leftarrow \text{A (test} = \text{false)}$$

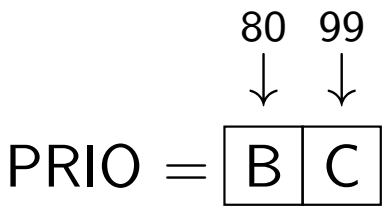
Se busca un estado objetivo siguiendo un camino hacia abajo en niveles. Se expande primero el nodo prioritario en la frontera.

Ventajas: Si se tiene suerte, se debe explorar solo una parte pequeña del espacio. Siempre encuentra la solución óptima.

Desventajas: Uso exponencial de recursos computacionales en el peor caso.

Algoritmo best first search



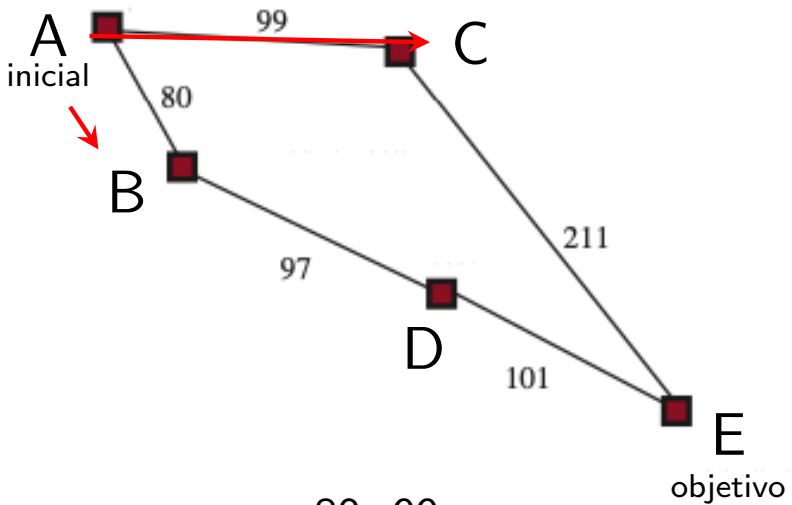


Se busca un estado objetivo siguiendo un camino hacia abajo en niveles. Se expande primero el nodo prioritario en la frontera.

Ventajas: Si se tiene suerte, se debe explorar solo una parte pequeña del espacio. Siempre encuentra la solución óptima.

Desventajas: Uso exponencial de recursos computacionales en el peor caso.

Algoritmo best first search



80 99

PRIO = | B | C |

nodo ← POP(Frontera)

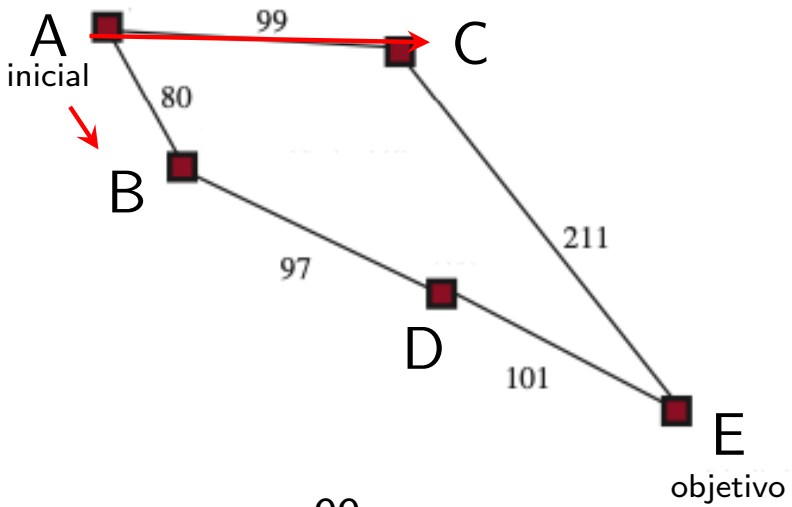
Se busca un estado objetivo siguiendo un camino hacia abajo en niveles. Se expande primero el nodo prioritario en la frontera.

Ventajas: Si se tiene suerte, se debe explorar solo una parte pequeña del espacio. Siempre encuentra la solución óptima.

Desventajas: Uso exponencial de recursos computacionales en el peor caso.

MACC
Matemáticas Aplicadas y Ciencias de la Computación

Algoritmo best first search



99
↓

PRIO = [C]

nodo ← B (test = false)

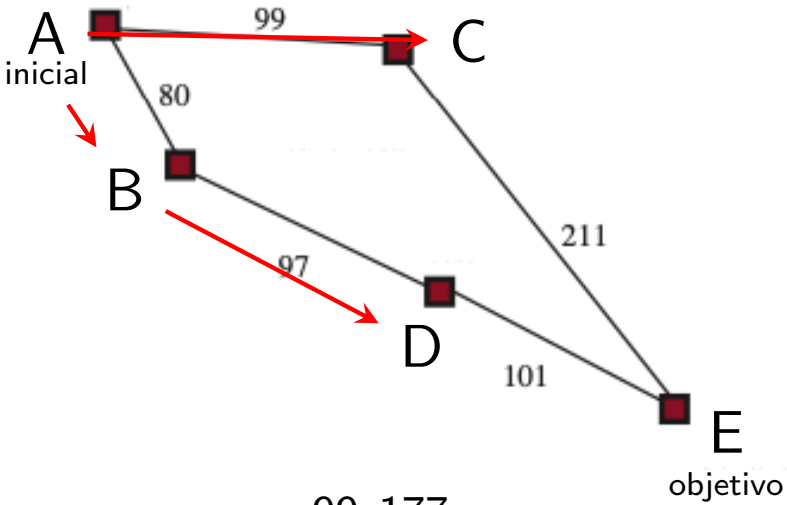
Se busca un estado objetivo siguiendo un camino hacia abajo en niveles. Se expande primero el nodo prioritario en la frontera.

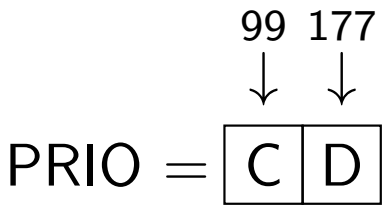
Ventajas: Si se tiene suerte, se debe explorar solo una parte pequeña del espacio. Siempre encuentra la solución óptima.

Desventajas: Uso exponencial de recursos computacionales en el peor caso.

MACC
Matemáticas Aplicadas y Ciencias de la Computación

Algoritmo best first search



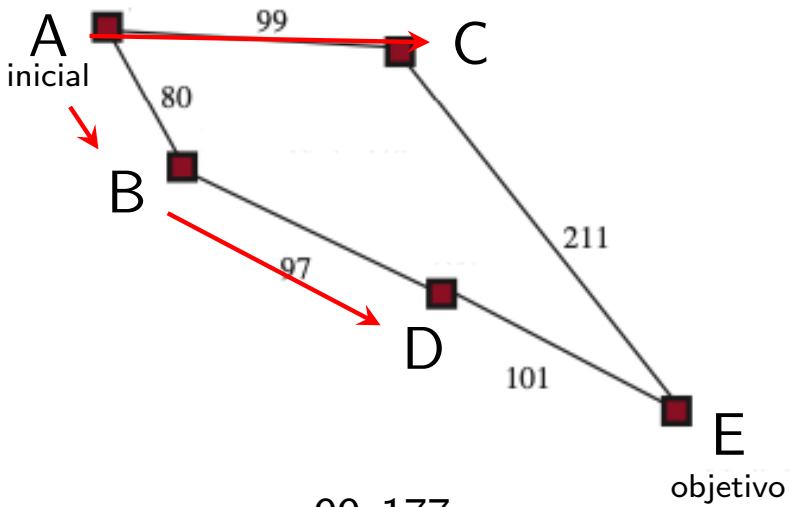


Se busca un estado objetivo siguiendo un camino hacia abajo en niveles. Se expande primero el nodo prioritario en la frontera.

Ventajas: Si se tiene suerte, se debe explorar solo una parte pequeña del espacio. Siempre encuentra la solución óptima.

Desventajas: Uso exponencial de recursos computacionales en el peor caso.

Algoritmo best first search



99 177

PRIO = | C | D |

nodo ← POP(Frontera)

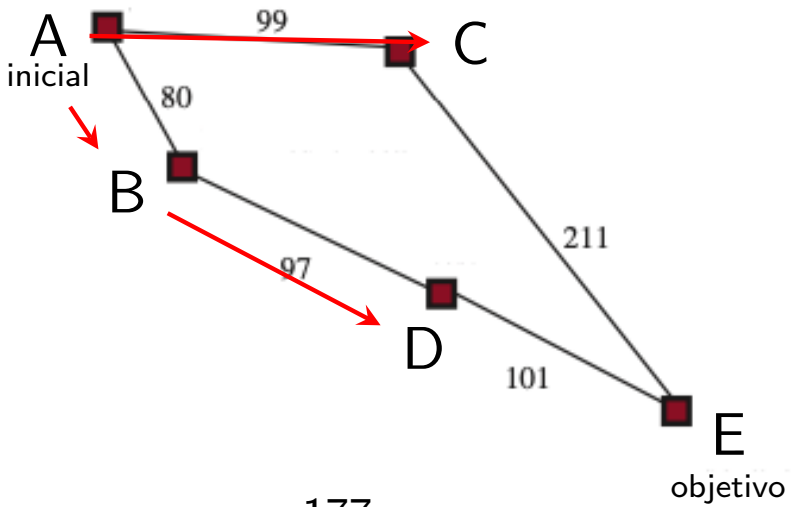
Se busca un estado objetivo siguiendo un camino hacia abajo en niveles. Se expande primero el nodo prioritario en la frontera.

Ventajas: Si se tiene suerte, se debe explorar solo una parte pequeña del espacio. Siempre encuentra la solución óptima.

Desventajas: Uso exponencial de recursos computacionales en el peor caso.

MACC
Matemáticas Aplicadas y Ciencias de la Computación

Algoritmo best first search



177

$$\text{PRIO} = \boxed{\text{D}}$$

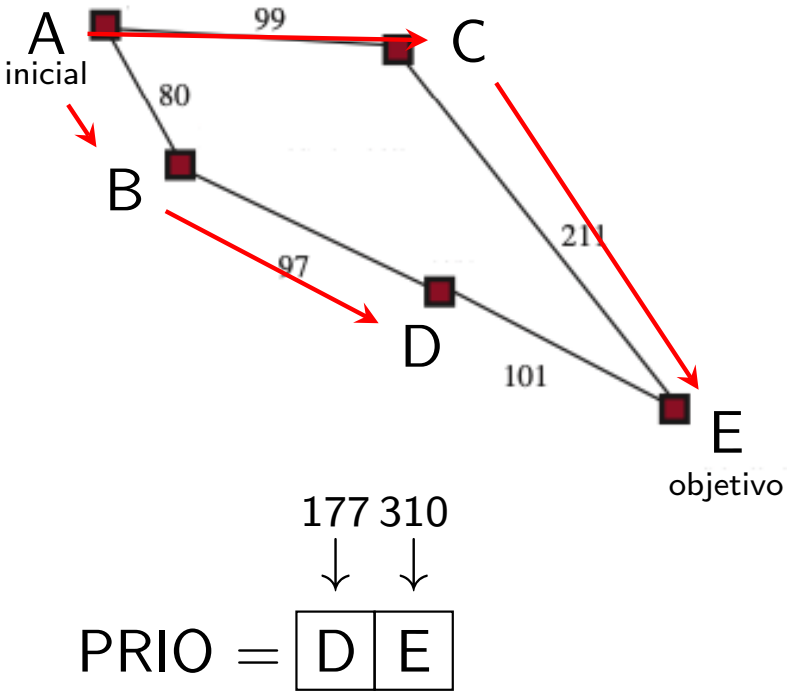
$$nodo \leftarrow \text{C (test} = \text{false)}$$

Se busca un estado objetivo siguiendo un camino hacia abajo en niveles. Se expande primero el nodo prioritario en la frontera.

Ventajas: Si se tiene suerte, se debe explorar solo una parte pequeña del espacio. Siempre encuentra la solución óptima.

Desventajas: Uso exponencial de recursos computacionales en el peor caso.

Algoritmo best first search

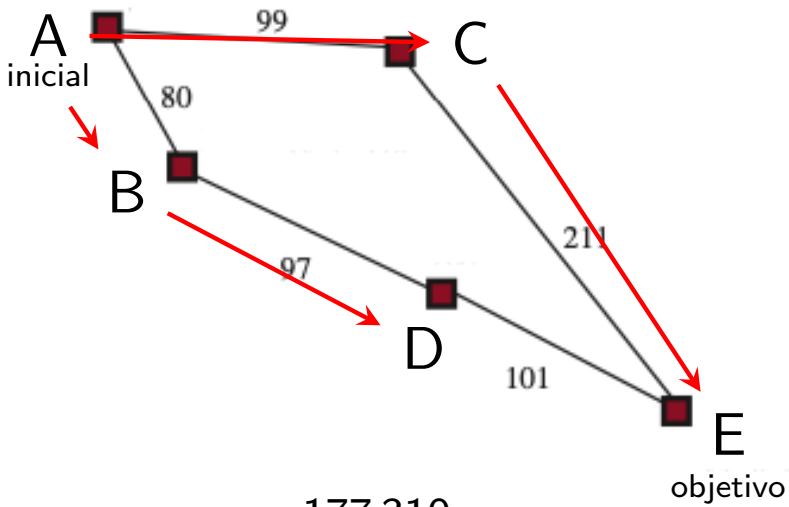


Se busca un estado objetivo siguiendo un camino hacia abajo en niveles. Se expande primero el nodo prioritario en la frontera.

Ventajas: Si se tiene suerte, se debe explorar solo una parte pequeña del espacio. Siempre encuentra la solución óptima.

Desventajas: Uso exponencial de recursos computacionales en el peor caso.

Algoritmo best first search



177 310

PRIO = | D | E |

nodo ← POP(Frontera)

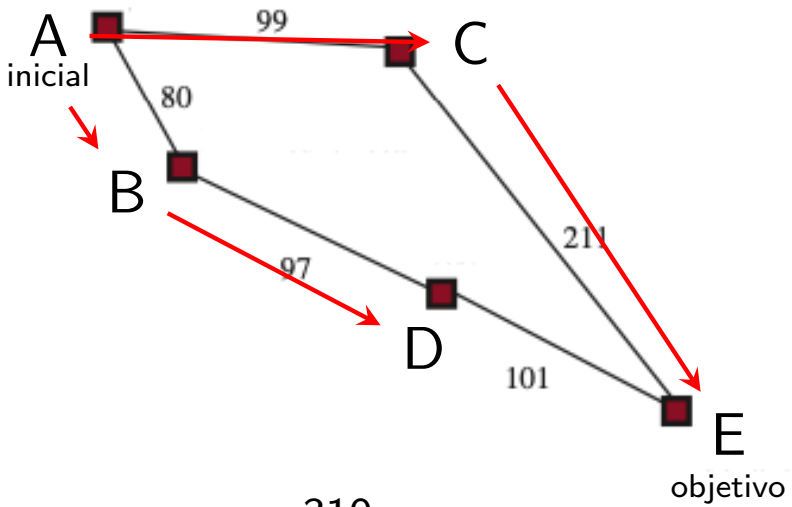
Se busca un estado objetivo siguiendo un camino hacia abajo en niveles. Se expande primero el nodo prioritario en la frontera.

Ventajas: Si se tiene suerte, se debe explorar solo una parte pequeña del espacio. Siempre encuentra la solución óptima.

Desventajas: Uso exponencial de recursos computacionales en el peor caso.

MACC
Matemáticas Aplicadas y Ciencias de la Computación

Algoritmo best first search



310

$$PRIO = \boxed{E}$$

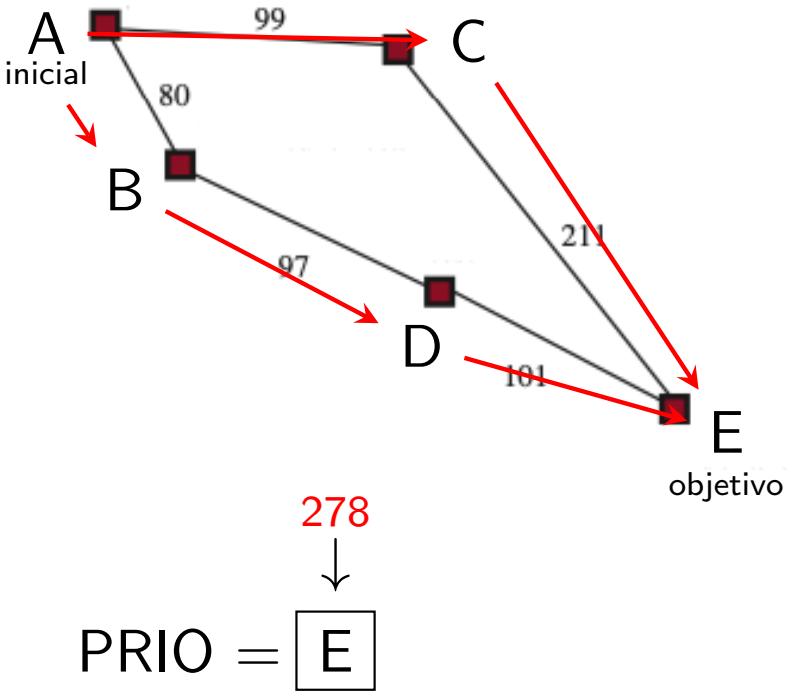
$nodo \leftarrow$ D (test = false)

Se busca un estado objetivo siguiendo un camino hacia abajo en niveles. Se expande primero el nodo prioritario en la frontera.

Ventajas: Si se tiene suerte, se debe explorar solo una parte pequeña del espacio. Siempre encuentra la solución óptima.

Desventajas: Uso exponencial de recursos computacionales en el peor caso.

Algoritmo best first search



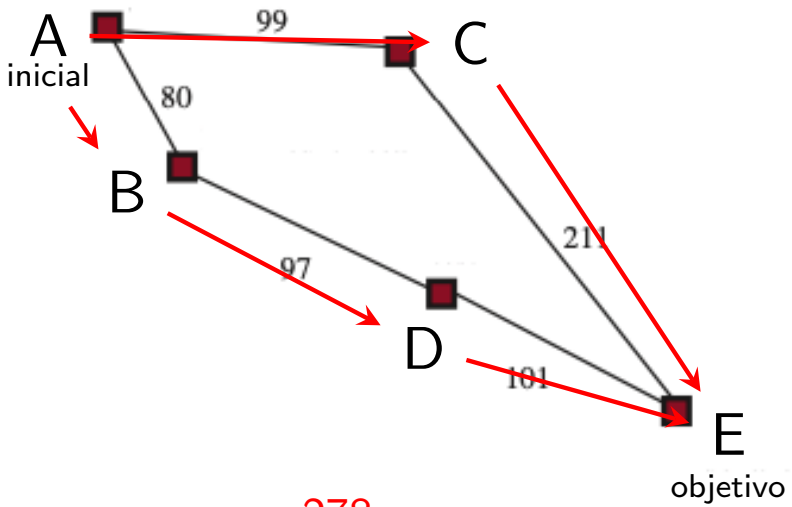
Se busca un estado objetivo siguiendo un camino hacia abajo en niveles. Se expande primero el nodo prioritario en la frontera.

Ventajas: Si se tiene suerte, se debe explorar solo una parte pequeña del espacio. Siempre encuentra la solución óptima.

Desventajas: Uso exponencial de recursos computacionales en el peor caso.

MACC
Matemáticas Aplicadas y Ciencias de la Computación

Algoritmo best first search



278

$$PRIO = \boxed{E}$$

$$nodo \leftarrow \text{POP}(\text{Frontera})$$

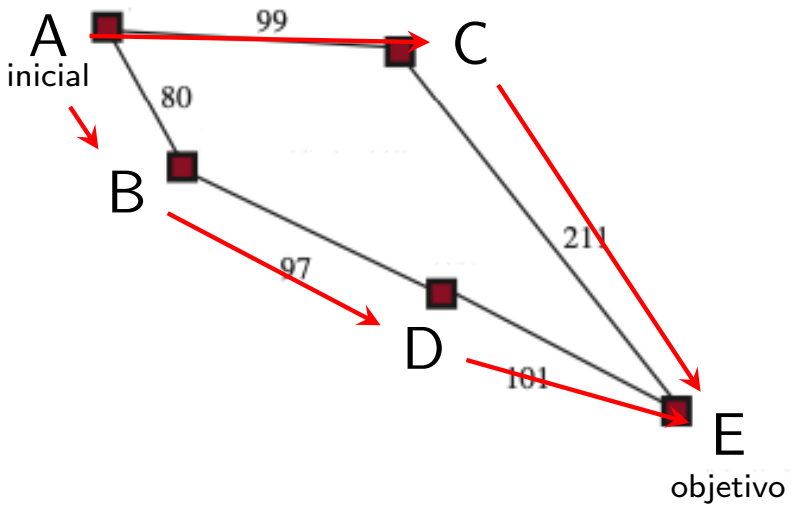
Se busca un estado objetivo siguiendo un camino hacia abajo en niveles. Se expande primero el nodo prioritario en la frontera.

Ventajas: Si se tiene suerte, se debe explorar solo una parte pequeña del espacio. Siempre encuentra la solución óptima.

Desventajas: Uso exponencial de recursos computacionales en el peor caso.

MACC
Matemáticas Aplicadas y Ciencias de la Computación

Algoritmo best first search



Se busca un estado objetivo siguiendo un camino hacia abajo en niveles. Se expande primero el nodo prioritario en la frontera.

Ventajas: Si se tiene suerte, se debe explorar solo una parte pequeña del espacio. Siempre encuentra la solución óptima.

Desventajas: Uso exponencial de recursos computacionales en el peor caso.

$$\text{PRIO} =$$

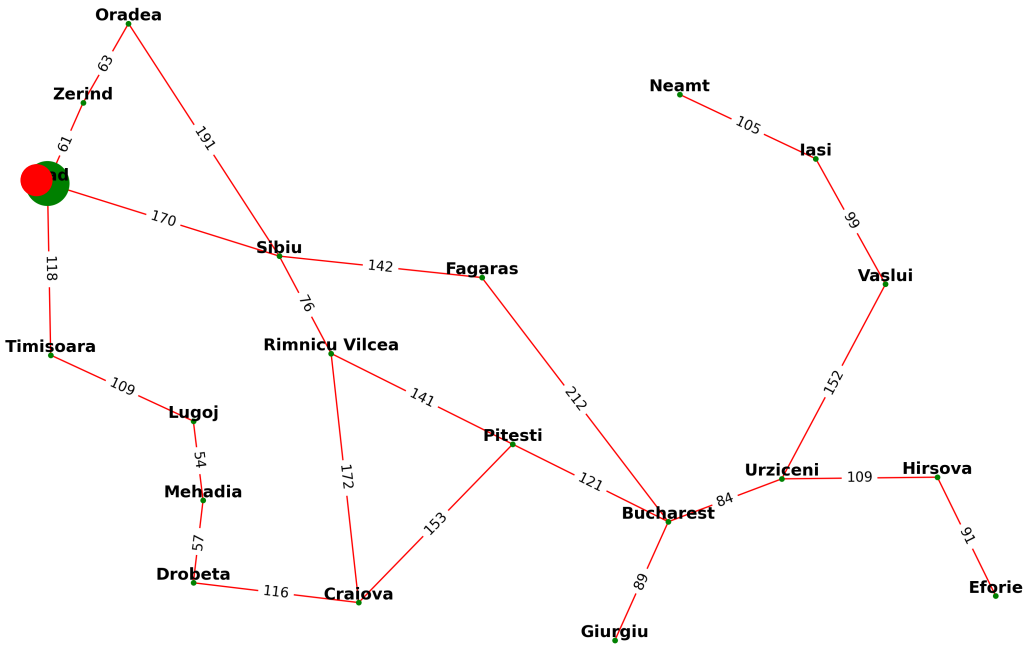
$$nodo \leftarrow \text{E (test} = \text{true)}$$

MACC
Matemáticas Aplicadas y Ciencias de la Computación

Costo camino (Dijkstra)

Problema: Ir de Arad a Bucharest.

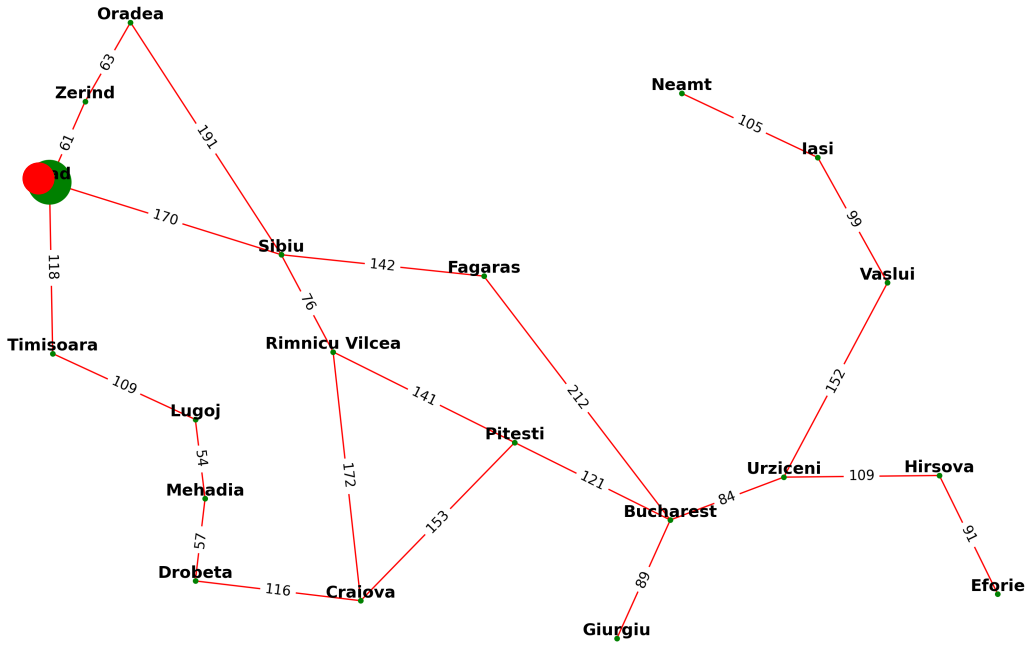
valor = costo_camino



Costo camino (Dijkstra)

Problema: Ir de Arad a Bucharest.

valor = costo_camino

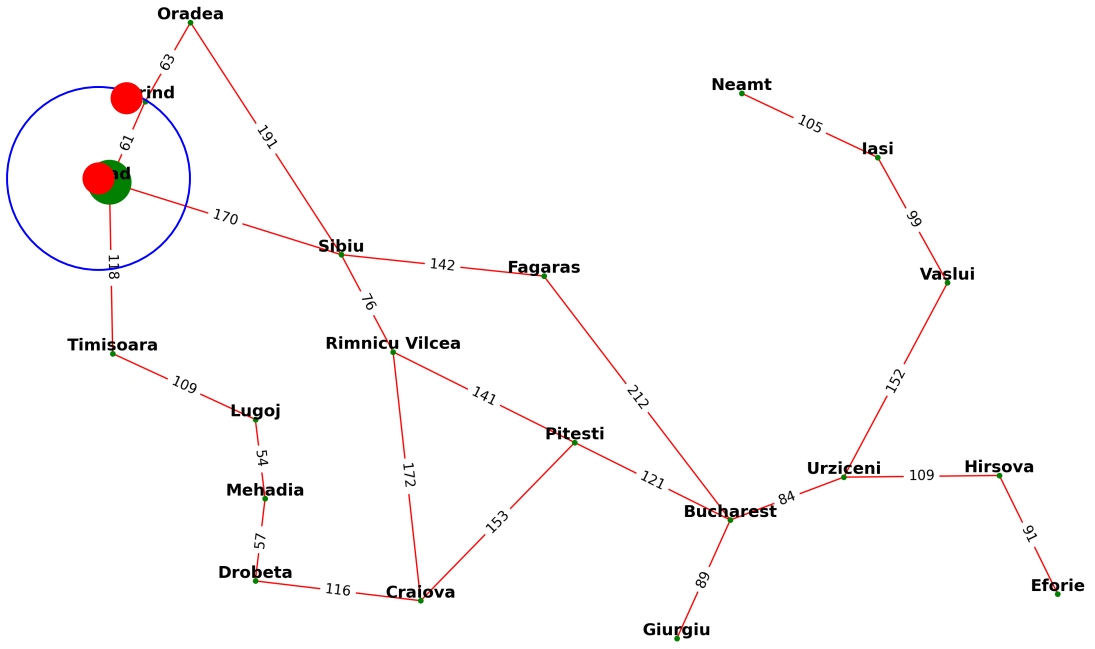


FRONTERA = $[(\text{Arad}_{\text{Zerind}}, 61), (\text{Arad}_{\text{Timisoara}}, 118), (\text{Arad}_{\text{Sibiu}}, 170)]$

Costo camino (Dijkstra)

Problema: Ir de Arad a Bucharest.

valor = costo_camino

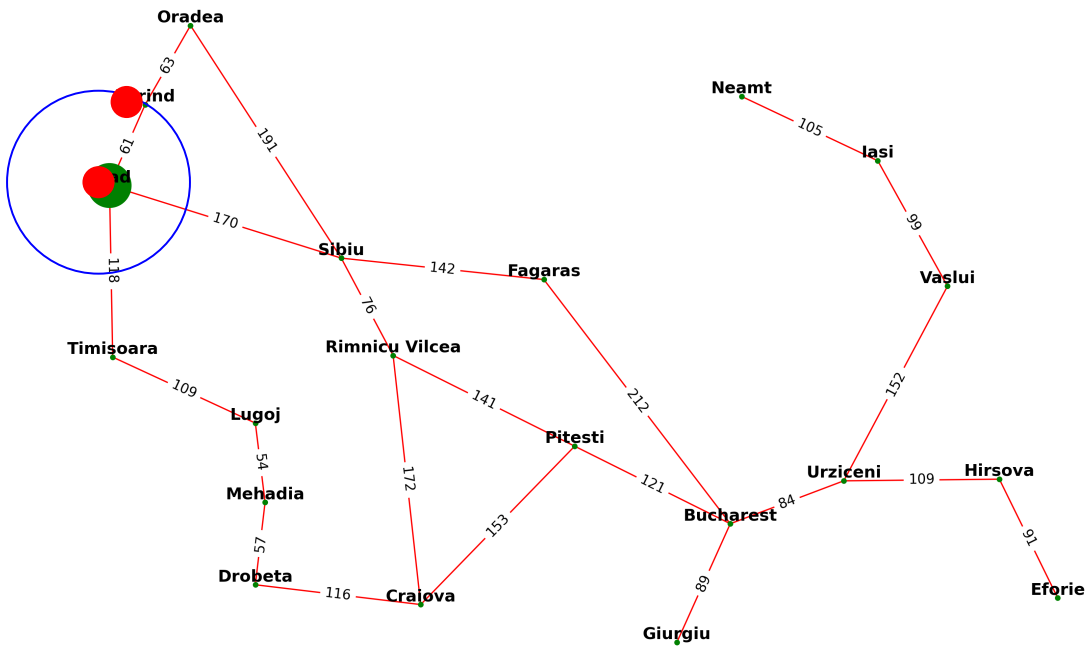


FRONTERA = $[(\text{Arad}_{\text{Zerind}}, 61), (\text{Arad}_{\text{Timisoara}}, 118), (\text{Arad}_{\text{Sibiu}}, 170)]$

Costo camino (Dijkstra)

Problema: Ir de Arad a Bucharest.

valor = costo_camino

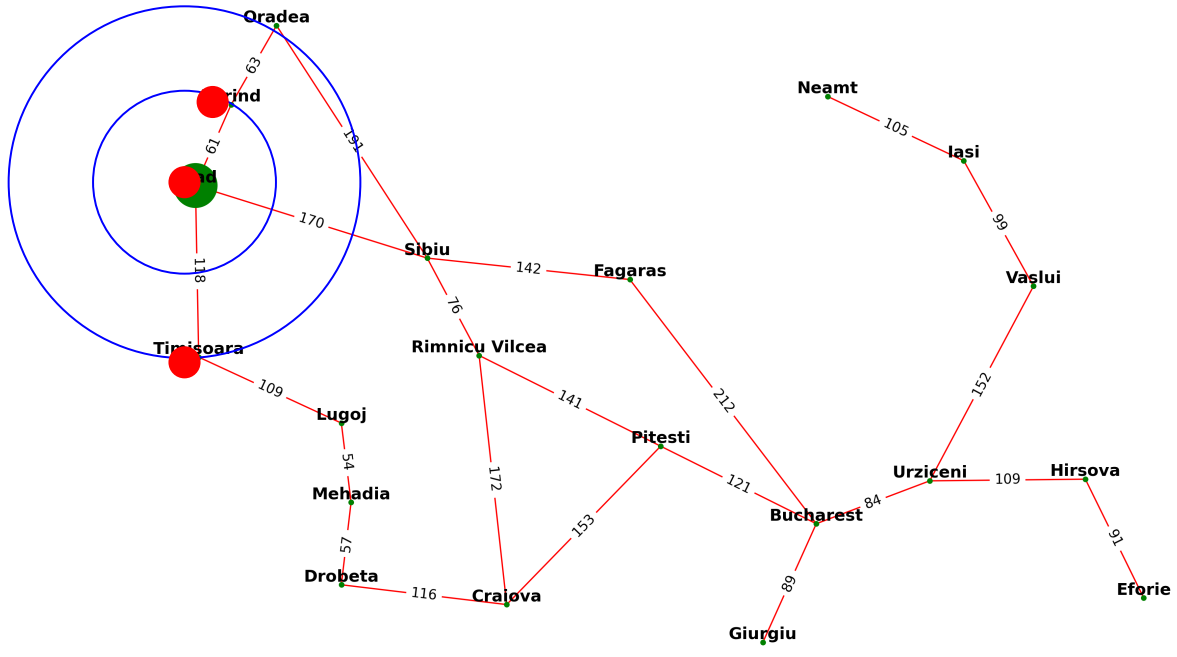


$$\text{FRONTERA} = [(\text{Arad}_{\text{Timisoara}}, 118), (\text{Arad}_{\text{Zerind}}^{\text{Oradea}}, 124), (\text{Arad}_{\text{Sibiu}}, 170)]$$

Costo camino (Dijkstra)

Problema: Ir de Arad a Bucharest.

valor = costo_camino

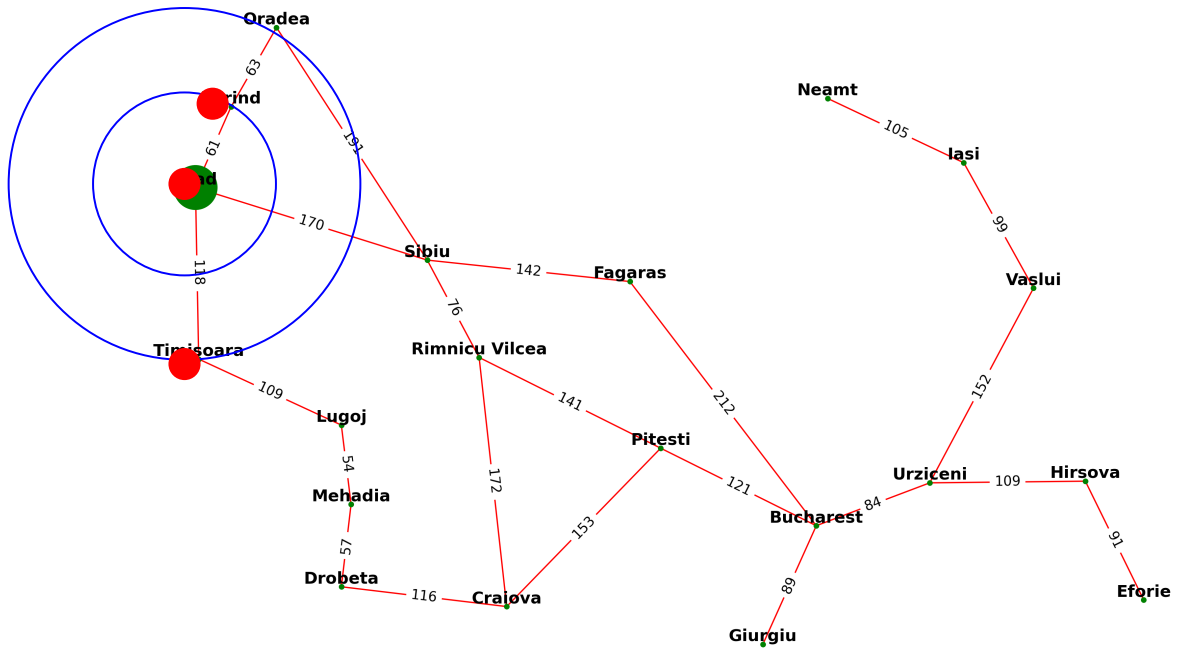


$$\text{FRONTERA} = [(\text{Arad Timisoara}, 118), (\text{Arad Zerind Oradea}, 124), (\text{Arad Sibiu}, 170)]$$

Costo camino (Dijkstra)

Problema: Ir de Arad a Bucharest.

valor = costo_camino

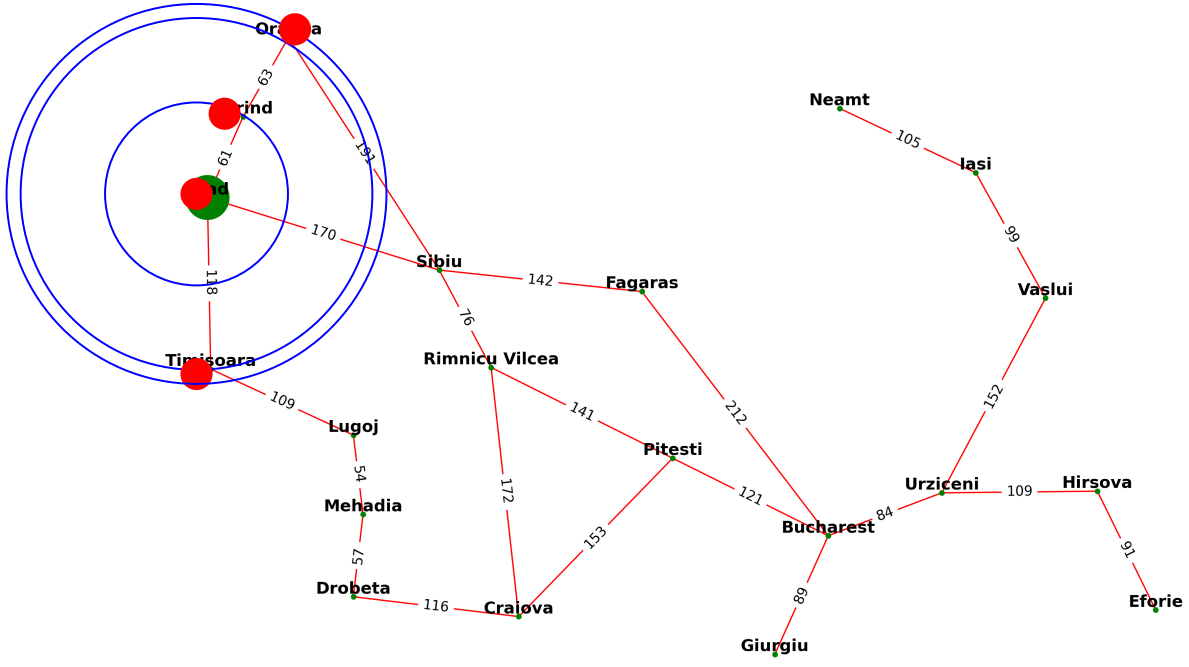


$$\text{FRONTERA} = [(\begin{matrix}\text{Arad}\\ \text{Zerind}\\ \text{Oradea}\end{matrix}, 124), (\begin{matrix}\text{Arad}\\ \text{Sibiu}\end{matrix}, 170), (\begin{matrix}\text{Arad}\\ \text{Timisoara}\\ \text{Lugoj}\end{matrix}, 227)]$$

Costo camino (Dijkstra)

Problema: Ir de Arad a Bucharest.

valor = costo_camino

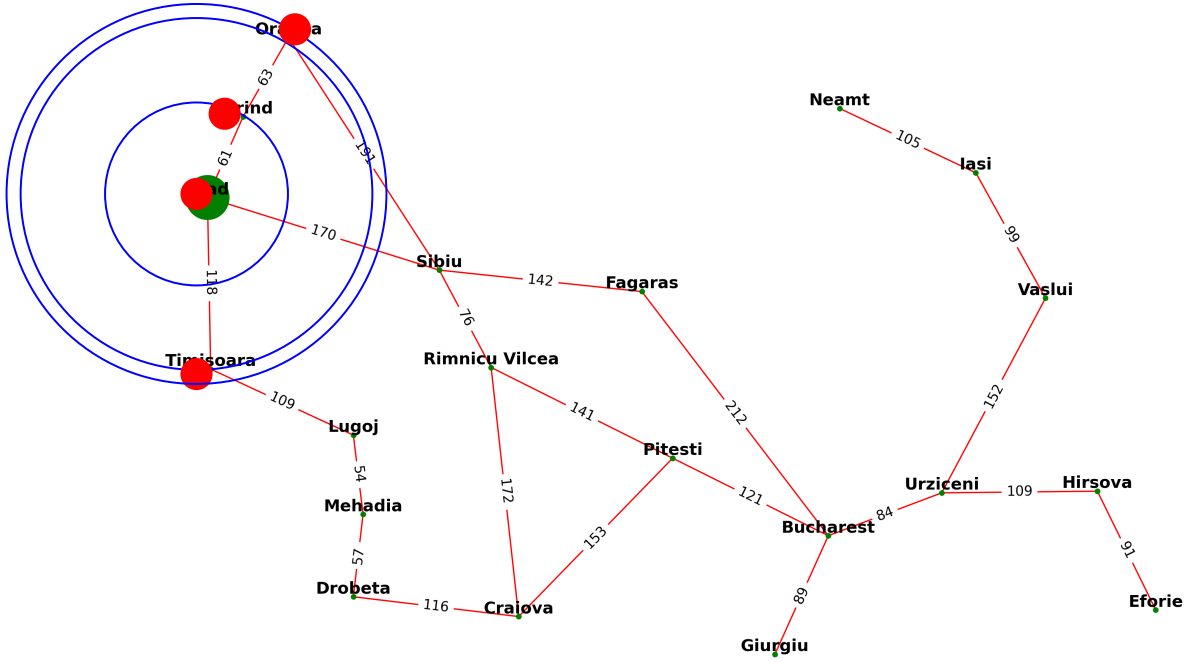


$$\text{FRONTERA} = [(\begin{matrix}\text{Arad}\\ \text{Zerind}\\ \text{Oradea}\end{matrix}, 124), (\begin{matrix}\text{Arad}\\ \text{Sibiu}\end{matrix}, 170), (\begin{matrix}\text{Arad}\\ \text{Timisoara}\\ \text{Lugoj}\end{matrix}, 227)]$$

Costo camino (Dijkstra)

valor = costo_camino

Problema: Ir de Arad a Bucharest.

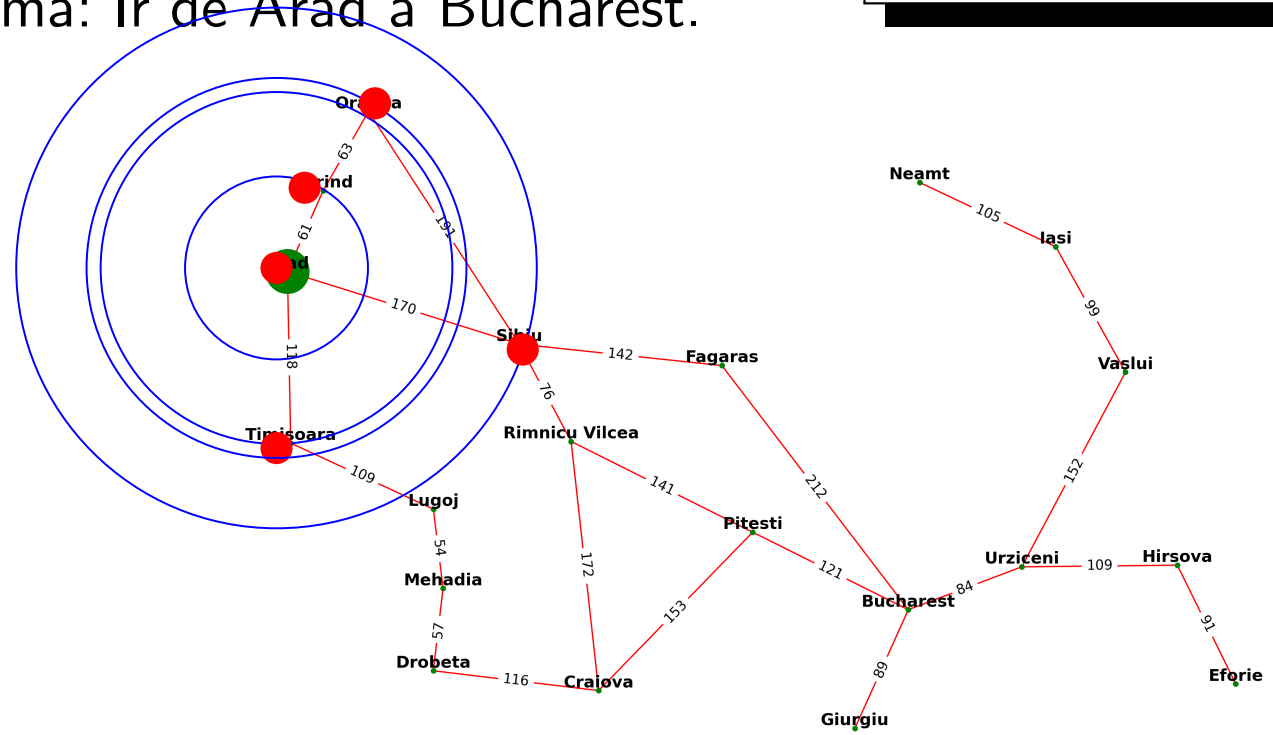


$$\text{FRONTERA} = [(\begin{matrix}\text{Arad}\\ \text{Sibiu}\end{matrix}, 170), (\begin{matrix}\text{Arad}\\ \text{Timisoara}\\ \text{Lugoj}\end{matrix}, 227), (\begin{matrix}\text{Arad}\\ \text{Zerind}\\ \text{Oradea}\\ \text{Sibiu}\end{matrix}, 315))]$$

Costo camino (Dijkstra)

valor = costo_camino

Problema: Ir de Arad a Bucharest.



$$\text{FRONTERA} = [(\begin{matrix}\text{Arad}\\ \text{Sibiu}\end{matrix}, 170), (\begin{matrix}\text{Arad}\\ \text{Timisoara}\\ \text{Lugoj}\end{matrix}, 227), (\begin{matrix}\text{Arad}\\ \text{Zerind}\\ \text{Oradea}\\ \text{Sibiu}\end{matrix}, 315))]$$

MACC
Matemáticas Aplicadas y Ciencias de la Computación

Take away

En esta sesión usted aprendió:

- Tipos de entornos de tarea.
- Idea general de la búsqueda en un especio de estados.
- Algunas estrategias de búsqueda a ciegas.
- El algoritmo general de búsqueda del mejor y su implementación sobre los costos (algoritmo de Dijkstra).