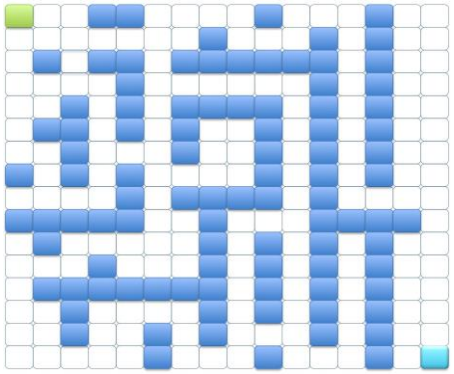
Grado en ingeniería informática
Inteligencia Artificial 2020/2021

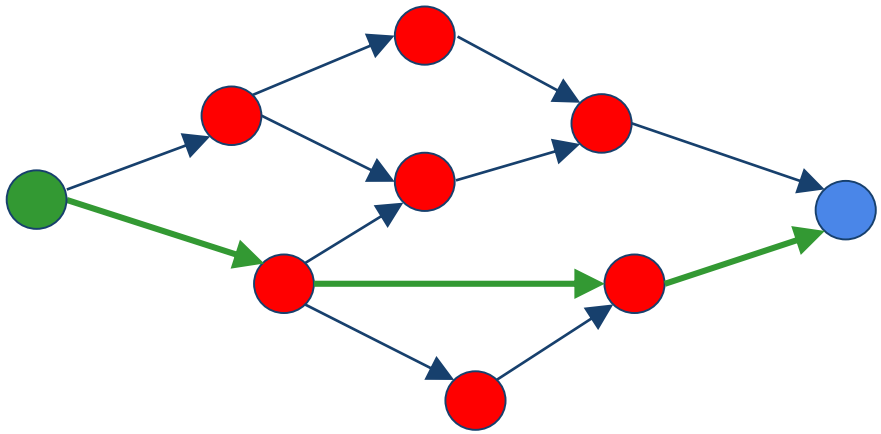
2. Resolución de problemas con búsqueda



UAM
Universidad Autónoma
de Madrid

Grado en ingeniería informática
Inteligencia Artificial 2020/2021

2.1. Búsqueda no informada



UAM
Universidad Autónoma
de Madrid

Lecturas :

- CAPÍTULO 3 de Russell & Norvig
- CAPÍTULOS 7 y 8 de Nilsson

Algunas figuras tomadas de http://aima.cs.berkeley.edu/

Tema 2: Resolución de problemas mediante búsqueda

1. Representación de problemas en un espacio de estados
2. Resolución de problemas con estrategias de búsqueda
   - Métodos no informados o ciegos
   - Métodos informados o heurísticos
   - Búsqueda entre adversarios

Búsqueda en un espacio de estados

Solucionar un problema mediante búsqueda:

Formulación + búsqueda + ejecución

- **Formulación del problema:**
  - Definir **estados**
  - Especificar el **estado inicial**
  - Especificar las **acciones** que puede realizar el agente
    - Reglas para las acciones permitidas.
    - Función sucesor:

      Estado actual → Lista de estados directamente accesibles.
  - Definir los estados **objetivo**
    - Definición extensiva: Lista
    - Definición intensiva: **Test de objetivo**
  - Definir **utilidad**: Función de coste del camino.

Camino: Secuencia de estados conectados por acciones.

Espacio de estados: Conjunto de estados accesibles desde el estado inicial

Se representa mediante un grafo conexo cuyos nodos ≡ estados, arcos ≡ acciones.

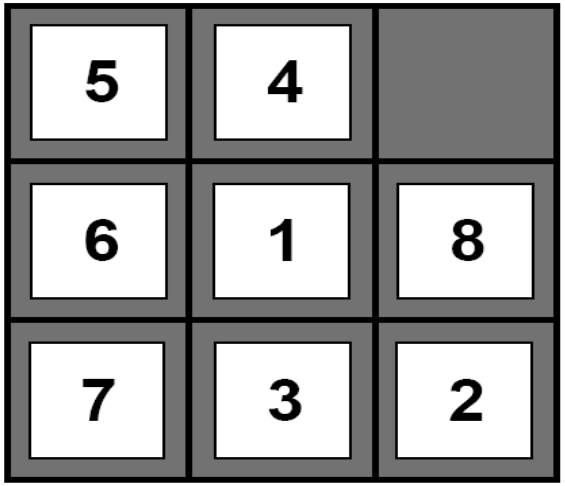
Solución: Camino desde el estado inicial a un estado objetivo.

Solución óptima: Solución con el mínimo coste.

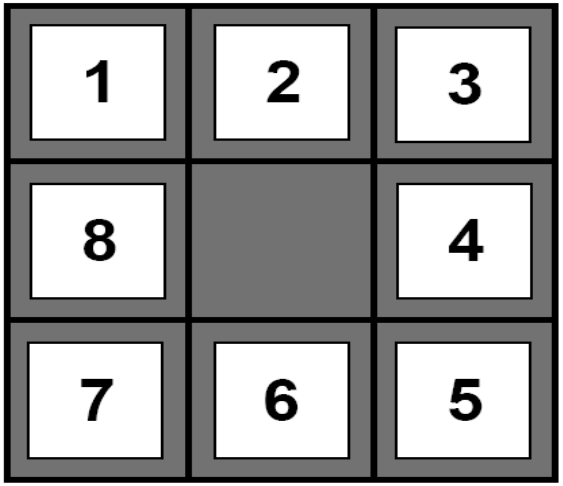
Problemas sencillos, I

8-puzzle.

- **Estados:** Disposiciones de 8 casillas numeradas de 1 a 8 + casilla vacía en un tablero de 3 x 3.
- **Estado inicial:** Una disposición dada (arbitraria).
- **Acciones:** Desplazar una pieza adyacente a la casilla vacía, a esa casilla.
  El hueco estará ahora en donde estaba la pieza que se ha movido.
- **Estado objetivo: Una disposición ordenada, con la casilla vacía en el medio.**
- **Utilidad**: Coste unidad por movimiento



Estado inicial

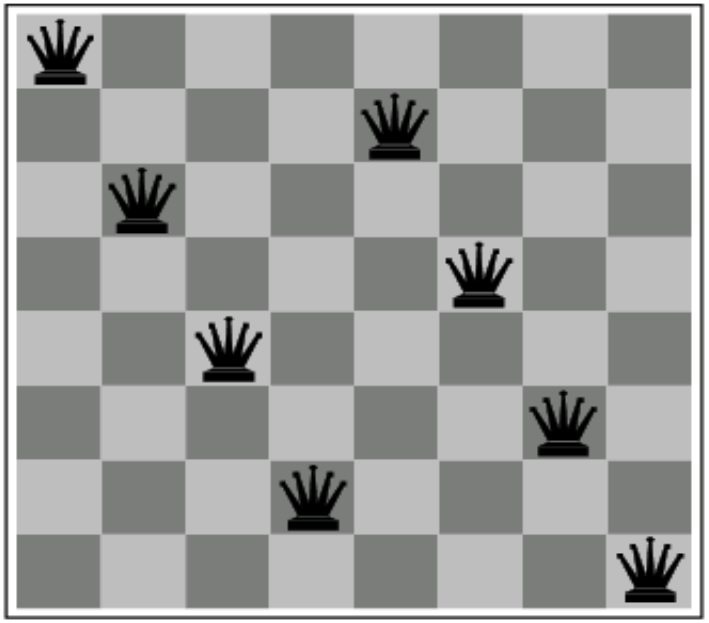


Estado objetivo

Problemas sencillos, II

Problema de las N reinas (formulación con estados completos)

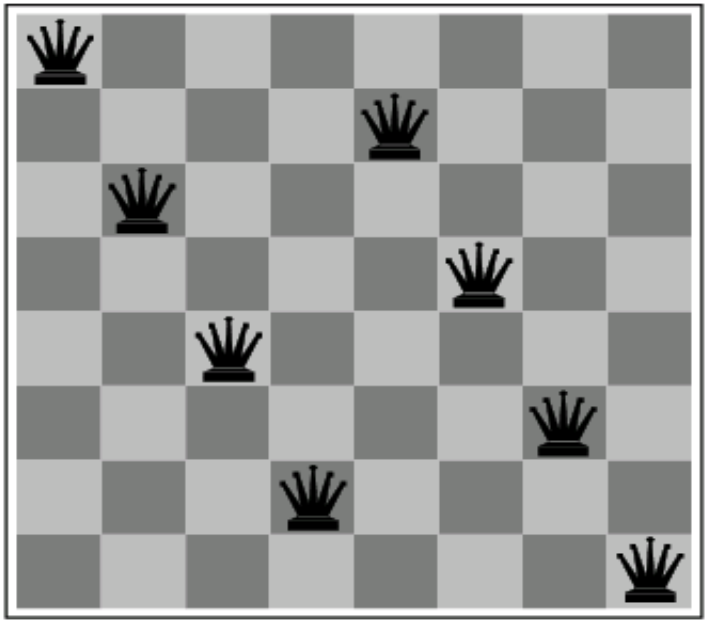
- **Estados:** Disposiciones de N reinas en un tablero de ajedrez de N x N casillas.
- **Estado inicial:** Una disposición dada (arbitraria) de las N reinas en el tablero.
- **Acciones:** Mover una reina a una casilla vacía.
- **Estado final:** N reinas que no se atacan entre ellas.
- **Utilidad**: Sólo es importante el estado final.



Problemas sencillos, III

Problema de las N reinas (formulación incremental)

- **Estados:** Disposiciones de n = 1,2, 3,… N reinas en un tablero de ajedrez de N x N.
- **Estado inicial:** Un tablero vacío.
- **Acciones:** A partir de un estado con n reinas, situar una reina nueva en una casilla vacía en la columna vacía más a la izquierda, que no ataque a las reinas previamente situadas.
- **Estado final:** N reinas que no se ataquen entre sí.
- **Utilidad**: Todas las soluciones tienen igual longitud (N pasos) e igual coste.



Problemas sencillos, IV

Criptoaritmética.

- **Estados:** Letras + dígitos.
- **Estado inicial:** Todas las letras.
- **Acciones:** Sustituir una letra por un dígito no usado.
- **Estado final:** Todas las letras se sustituyen por dígitos de tal forma que la aritmética es correcta.
- **Utilidad**: Todas las soluciones tienen igual longitud (10 pasos) e igual coste

```
    FORTY                29786
+     TEN            +     850
      TEN                  850
   ------               ------
    SIXTY                31486
```

Problemas reales I

- Buscador de rutas (por ej. viaje en avión)
  - **Estados:** Aeropuerto + hora de llegada.
  - **Estado inicial:** Aeropuerto de salida + hora de partida
  - **Acciones:** Estados resultantes de tomar uno de los vuelos programados disponibles desde el aeropuerto actual, saliendo más tarde que la hora actual + tiempo de tránsito del aeropuerto.
  - **Estado final:** Aeropuerto destino + hora de llegada deseada.
  - **Utilidad**: Coste económico, tiempo total de viaje, número de transbordos, confortabilidad, etc.

- Problemas de planificación de rutas:
  - **PVC** : Problema del viajante de comercio.

    Problema de diseño óptimo de tours: Encontrar un recorrido a través de N ciudades dadas en el que cada ciudad se visite sólo una vez, minimizando la longitud del camino.
    - **Estados:** N ciudades.
    - **Estado inicial:** Ciudad de partida.
    - **Acciones:** Viajar a una ciudad aún no visitada que sea directamente accesible por carretera desde la ciudad actual.
    - **Estado final:** Ciudad de partida, habiendo visitado exactamente una vez cada una de las otras ciudades
    - **Utilidad**: Distancia total recorrida.

Problemas reales II

- Diseño de distribución de componentes en chips VLSI
  - Posicionar componentes y conexiones en un chip para:
    - Minimizar: área, latencias, capacitancias no deseadas, generación de calor, costes.
    - Maximizar: nivel de producción.
- Navegación de robots
- Problemas de ensamblaje secuencial en fábricas (razonamiento espacial)
- Diseño de proteínas (predecir la estructura terciaria a partir de la estructura primaria).
- Búsqueda en Internet

Espacio de Estados: Conceptos básicos

- Dado un problema, necesito representarlo en el ordenador usando:
  - Estado: situación actual, conjunto de todas las características relevantes
    - Ajedrez: situación de las piezas en el tablero
    - Ruta de tráfico: densidad de tráfico en cada calle de la ciudad
- ¿Cómo alcanzar la solución de un problema?
  - Ejecución de acciones válidas (movimientos usando operadores)
    - Ajedrez: muevo una pieza
    - Ruta tráfico: escojo ir por una calle
  - ¿Qué ha cambiado? otra situación diferente (otro estado)
    - Ajedrez: otra situación de las piezas
    - Ruta de Tráfico: he avanzado por esa calle, estoy en otro lugar
- Espacio de Estados: El conjunto de todas las situaciones posibles
- Solución: un estado o el camino para llegar a la meta
- Llegar a las solución depende de qué movimientos haga el agente
  - ¿Qué estrategia de búsqueda utilizo?

Cómo definir el Estado en un problema

- Características relevantes. Sí, pero…¿cuáles lo son?
  Cuando modificamos, mediante una acción, el estado del sistema,
  ¿en qué influye que sea relevante para la solución del problema?

- ¿Dónde termina un estado y empieza el siguiente?

  Ej. Trayectoria de una nave desde la Tierra a Marte.
  - Un estado es una *foto* en un instante hecha cada cierto intervalo de tiempo.
  - Se hace una estimación aproximada de la localización.

- ¿Qué ocurre si cambian varias cosas a la vez?

  Ej. partido de tenis: ¡los dos jugadores se mueven a la vez!
  - Infinitos estados, cambios continuos
  - Localización aproximada
  - tiempo continuo pasa a discreto (el proceso es monitorizado en instantes discretos)

  Ej: partido fútbol: mucha más complejidad

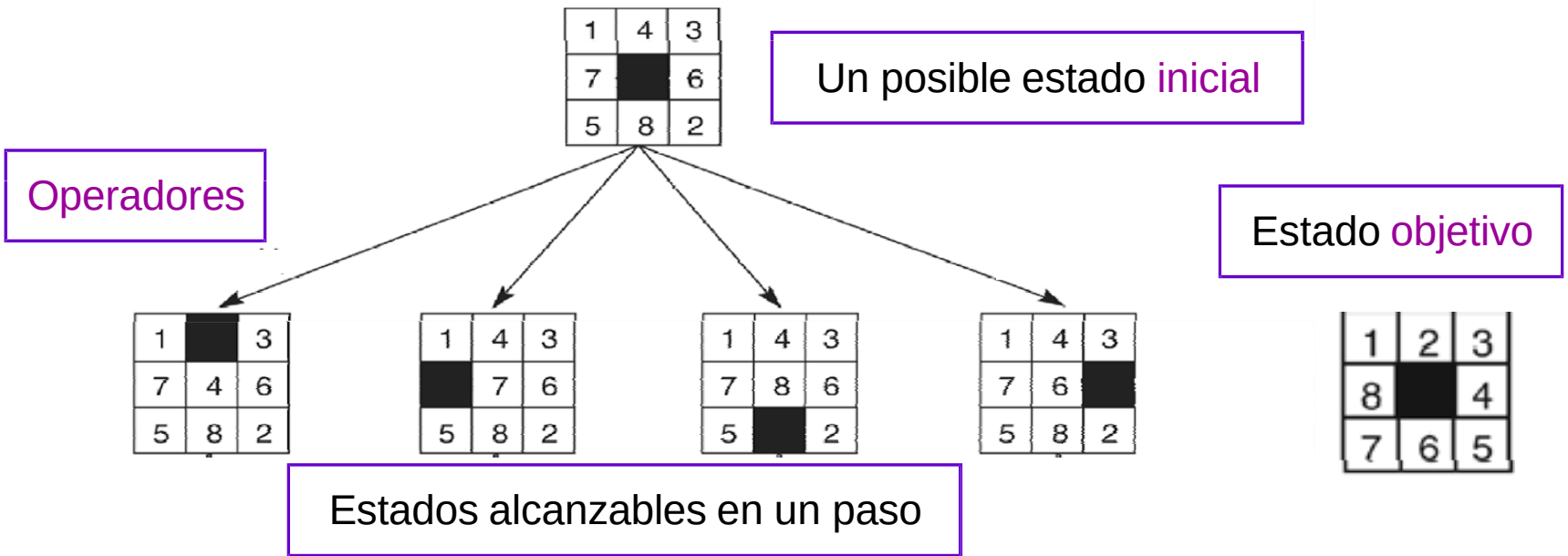
- ¿Qué ocurre si desconocemos partes relevantes del problema?
  - Heurísticas: Estimaciones del coste a partir de aproximaciones del problema mediante problemas similares cuya solución es conocida
  - Ignorar la información: la búsqueda funciona peor. Puede no o no encontrará solución

Convenio de representación: Elementos y Pasos

1. **Estado:** definir elementos que caracterizan un estado
2. **Estado inicial**
3. **Estados objetivo:** definir las condiciones de debe cumplir un estado para determinar si se ha alcanzado el objetivo.
4. **Operadores** (o acciones o función que genera sucesores o movimiento):
   - Acciones disponibles para transformar un estado en el siguiente
   - Se definen pre- y postcondiciones para los estados inicial (previo a la acción) y final (posterior a la acción), respectivamente
   - Coste del operador : representa el esfuerzo de aplicar dicho operador una vez
5. **Solución**: camino desde el estado inicial a un estado objetivo
   - Coste de la solución: Suma del coste de los operadores aplicados desde el estado inicial hasta el estado objetivo
   - Pueden haber soluciones (caminos) de diferentes costes
   - **Solución óptima: Solución de menor coste**

Ejemplo del 8-puzzle

- Estado
  - Tablero de 3*3
  - 8 fichas numeradas
  - 1 hueco
- Definición del espacio de estados
  - Estados alcanzables desde el estado inicial
  - Definido implícitamente por el estado inicial y los operadores
  - Grafo dirigido: vértices (estados), enlaces (acciones)



Representar Espacio de Estados: Abstracción adecuada

- Se representa un problema mediante abstracción
  - Eliminar detalles irrelevantes en la representación
- Una representación puede simplificar o complicar la resolución del problema
- Ej. Representación del problema del 8-puzzle → (¿es útil para resolver el problema?)
  - El estado no debe incluir información irrelevante para la solución del problema; por ejemplo, el material o el color del tablero .
  - Estados: localización de cada ficha (y hueco) en cada una de las 9 casillas en el tablero.
- Estado inicial: disposición inicial de las fichas en el tablero
- Coste de operadores: En este ejemplo, suponemos que un operador tiene coste 1
- Coste del camino: suma de los costes de los operadores aplicados = número de pasos dados.

Espacio de Estados y operadores: Niveles de abstracción

- Estados y operadores deben definirse de manera que sus definiciones sean compatibles y complementarios, de forma faciliten la resolución del problema.
- Los operadores deben ser
  - Deterministas: aplicados al mismo estado dan siempre el mismo resultado
  - Tan generales como sea posible: parametrizarlos
  - Reducir el número de reglas distintas (operadores).

Ejemplo del 8-puzzle

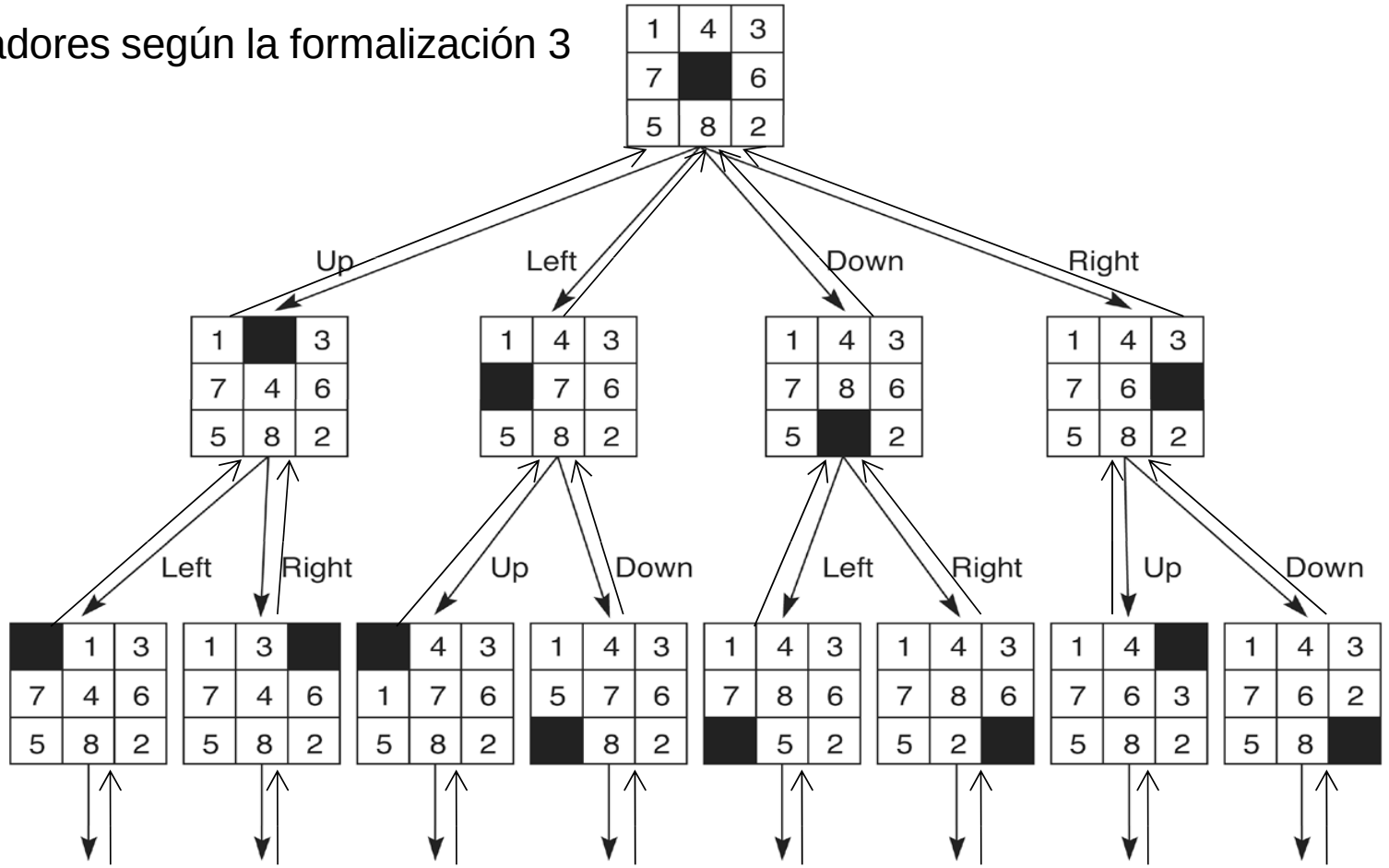
- Estados: Disposiciones de las 8 fichas numeradas y el hueco: 9! estados distintos
- Hay distintas formas de definir los operados
  1. Lista de 9! estados y estados sucesores: máximo 9!*4 *-- demasiado específicos*
  2. move (<ficha>, <dirección>): 8*4 operadores *-- preferible, pero puede mejorarse*
     <ficha> ∈ {1, 2,…, 8}; <dirección> ∈ {up, down, left, right}
  3. move_empty(<dirección>): 4 operadores *-- óptimo*

     <dirección> ∈ {up, down, left, right}

     4 operadores para desplazar el hueco arriba, abajo, derecha o izquierda

Árbol de búsqueda para el 8-puzzle

Operadores según la formalización 3



Representación del problema: descripción e implementación

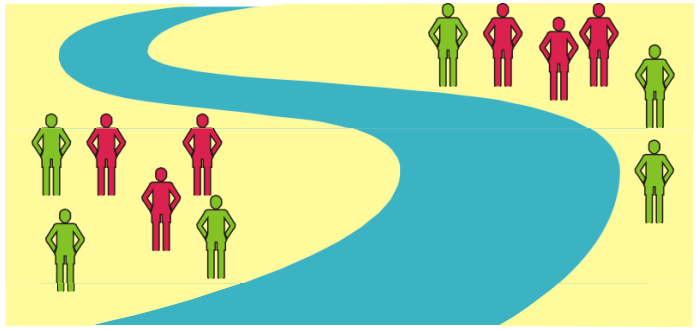
- **Descripción** de la representación:
  - Especifica estados y operadores
  - Mediante diagramas, pseudocódigo, etc.
- **Implementación** de la representación:

  Para resolver el problema, estados y operadores son representados mediante un lenguaje de formal (ej. un lenguaje de programación)
  - Los estados se implementan mediante una estructura de datos
  - Los operadores se implementan mediante operaciones (funciones) en dicho lenguaje formal

- Ejemplo: Representación de los estados del 8-puzzle
  - Descripción
    - Estados: localización de cada ficha y del hueco en cada una de las 9 casillas
  - Implementación: diversas opciones
    - Matriz 3*3,
    - Vector de longitud 9,
    - Conjunto de hechos {(superior_izda = 3), (superior_centro = 8), ...}

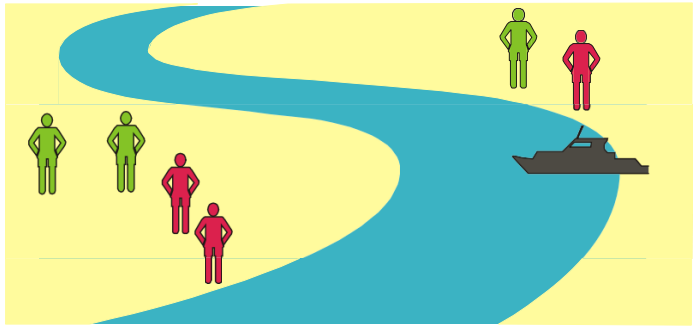
Ejemplo: los misioneros y los caníbales

- Enunciado
  - 3 misioneros y 3 caníbales en la orilla de un río junto con 1 bote
  - El objetivo es que pasen todos a la otra orilla
  - Hay dos restricciones
    - Deben cruzar usando el bote en el que sólo pueden ir 1 o 2 personas
    - En caso de que haya misioneros, en ninguna de las orillas o e el bote puede haber más caníbales que misioneros



- Representar el problema según el paradigma del espacio de estados y dibujar el espacio de estados

Ejemplo: misioneros y caníbales



Hay un total de 7 objetos → ¿guardamos la posición de cada uno de ellos?

1. Identificar a las personas concretas: *(m1, m2, m3, c1, c2, c3, b)*

   Estado = (1, 1, 0, 1, 0 ,1, 1) [0: orilla derecha, 1: orilla izquierda]

2. Es preferible abstraer y dejar fuera características irrelevantes como la identidad concreta de cada uno de los misioneros y caníbales: indicar únicamente el nº de misioneros, caníbales en cada orilla y posición del bote

   (nm_oi, nc_oi, nm_od, nc_od, b) *[nm_oi + nm_od = 3, nc_oi + nc_od = 3]*

   Estado = (2, 2, 1, 1, 0)

3. Teniendo en cuenta que hay un total de 3 misioneros y 3 caníbales, podemos representar únicamente el número de misioneros y caníbales en la orilla izda.

   (nm_oi, nc_oi, b) *[nm_od = 3 - nm_oi; nc_od = 3 - nc_oi]*

   Estado = (2, 2, 0)

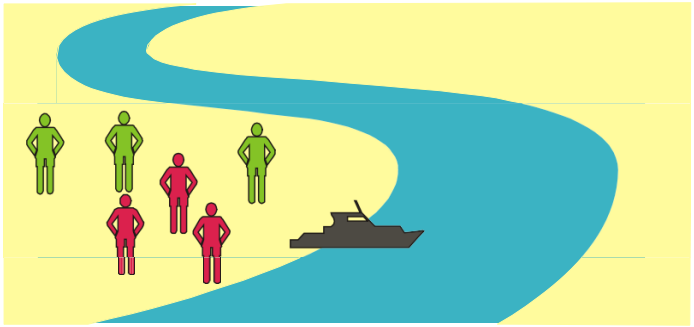
Misioneros y caníbales: estados

- Descripción de la Representación **`PASO 1: definir el estado`**
  - **Estado** = nº de misioneros, caníbales y bote en la orilla de partida
  - Suponemos:
    - La orilla inicial es la orilla izquierda del río
    - El cruce del río se produce en un paso (el tiempo que se invierte en cruzar el río es irrelevante para la solución del problema)
  - Estado = *(nm, nc, b)*
    - *nm_oi:* número de misioneros en la orilla izquierda *(0, 1, 2 o 3)*
    - *nc_oi:* número de caníbales en la orilla izquierda *(0, 1, 2 o 3)*
    - *b:* posición del bote *(0 = orilla derecha ó 1 = orilla izquierda)*

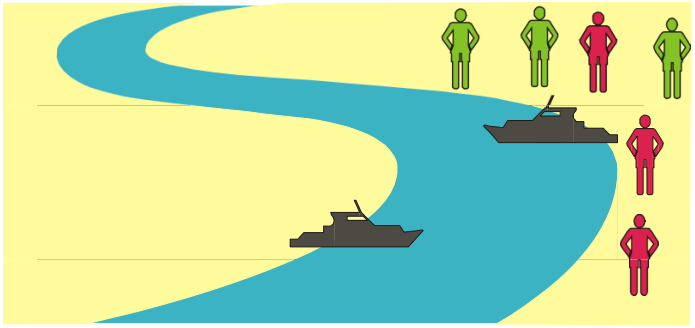
- La orilla en la que se encuentra el bote es importante. En concreto, se utiliza para determinar determina si se pueden aplicar o no los operadores
  - *(2, 1, 0) ≠ (2, 1, 1)*

Misioneros y caníbales: estado inicial y objetivo

PASO 2: definir estados inicial y objetivo

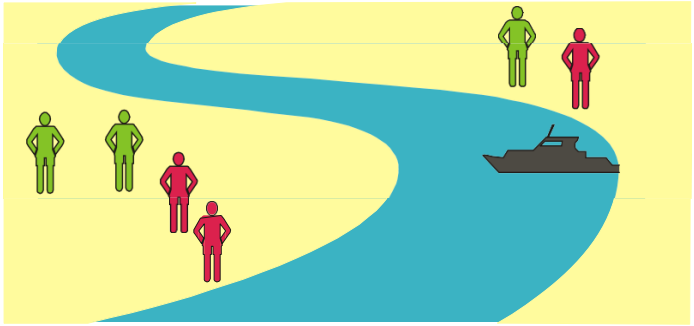


Estado inicial (3, 3, 1)



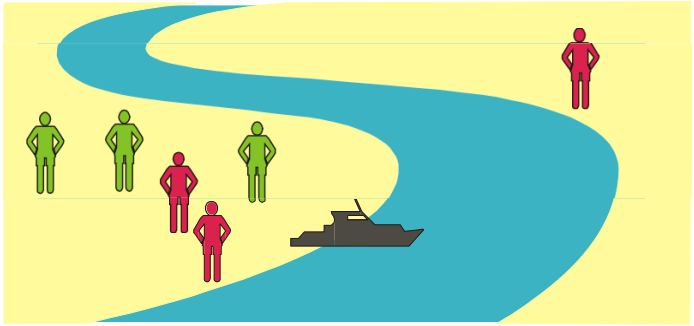
Estado objetivo (0, 0, 0)

(0, 0, 1) no es posible



(2, 2, 0)

Estados intermedios posibles



(3, 2, 1)

Misioneros y caníbales: acciones

Operadores: *¿Qué determina un cambio de estado?* **(PASO 3)**

Hay 5: el bote siempre cruza el río junto a 1 o 2 personas

- *cruzaM:* Cruzar con 1 misionero.
- *cruzaMM:* Cruzar con 2 misioneros.
- *cruzaC:* Cruzar con 1 caníbal.
- *cruzaCC:* Cruzar con 2 caníbales.
- *cruzaMC* : Cruzar con 1 misionero y 1 caníbal

- Coste: asumimos que el coste operador = 1
  Coste de camino = número de veces que se cruza el río

- Restricciones
  - Precondiciones: La orilla en la que se encuentra el bote es importante
    Ej ., no podría cruzar ningún misionero en los estados (0, _, 0) y (3, _, 1)
  - Postcondiciones: En caso de que haya misioneros, el número de caníbales no puede superar al de misioneros en ninguna de las dos orillas.

- c*ruzaM(nm_oi, nm_od, B)*
  Precondiciones: { ($nm\_oi > 0 \wedge b = 0$) $\vee$ ($nm\_od < 3 \wedge b = 1$) }
  si *b = 1* entonces *nm_oi := nm_oi -1* $\wedge$ *b := 0* → *(nm_oi -1, nc_oi, 0)*
  si *b = 0* entonces *nm_oi := nm_oi +1* $\wedge$ *b := 1* → *(nm_oi +1, nc_oi, 0)*
  Postcondinción: El estado final debe ser un estado permitido.

Misioneros y caníbales: restricciones

Situaciones no permitidas: En caso de que haya misioneros, en ninguna de las orillas o e el bote puede haber más caníbales que misioneros
¿en bote? ¿en orillas?

Definiremos los operadores de forma que estas situaciones no se puedan dar.

- **En el bote**: Viajan un máximo de 2 personas, por lo que no puede haber más caníbales que misioneros en él.
  Si el máximo hubiera dido fuese mayor que 2, habría situaciones no permitidas *(y sería otro problema)*

- **En las orillas**: Dado el estado *(nm_oi, nc_oi, b)*
  - *(3, 3, 1)* y *(2, 2, 0)* son estados permitidos
  - *(1, 2, 0)* y *(2, 3, 0)* son estados no permitidos
  - ¿Bastará $nm\_oi < nc\_oi$ como condición de estado no permitido?
    - En *(2, 1, 0)* ¿es posible?
      - ¡En la orilla derecha hay 1 misionero con 2 caníbales!
    - En *(0, 2, 1)* ¿es posible?
      - ¡Si no hay misioneros el estado es posible!

- **Condición para estados no permitidos:**
  $(nm\_oi < nc\_oi \wedge nm\_oi \neq 0) \vee (nm\_oi > nc\_oi \wedge nm\_oi \neq 3)$

Misioneros y caníbales: Espacio de estados

Tamaño del espacio de estados

(nm_oi, nc_oi, b)

nm_oi $\in$ *{0,1,2,3}, nc_oi* $\in$ *{0,1,2,3}; b* $\in$ *{0,1}*

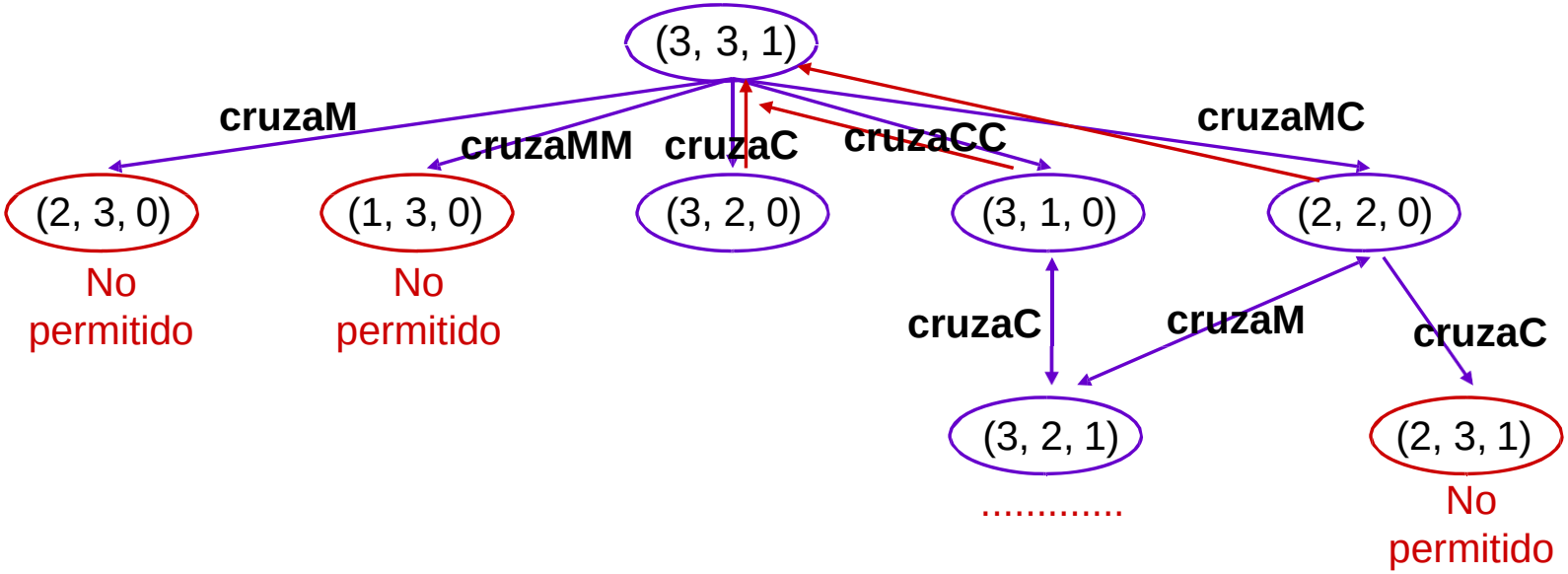
- Máximo: 4*4*2 = 32 estados posibles
  - Hay 4 estados inalcanzables (por lo tanto, hay 28 estados alcanzables)
    - Obvios como (0, 0, 1) y (3, 3, 0)
    - Y no tan obvios como (3, 0, 1) y (0, 3, 0)
  - Hay 12 estados no permitidos
    - (1, 2, _), (2, 3, _), (1, 3, _), (2, 1, _), (2, 0, _), (1, 0, _)

  Espacio de estados:16 estados alcanzables permitidos
- A veces la condición de estado no permitido no se incluye como postcondición en el operador, sino que se comprueba con posterioridad en el algoritmo de búsqueda.

  En ese caso, el espacio de estados se compone de 12+16 = 28 estados.

Misioneros y caníbales: Grafo

Espacio de estados representado como un grafo dirigido (puede haber ciclos)



- Desarrollad vosotros el resto del espacio de estados a partir del estado (3, 2, 1)

Misioneros y caníbales: Resumen

Representación formal del problema

- Estado inicial: **(3, 3, 1)**. Estado objetivo: **(0, 0, 0)**
- Restricciones: estados no válidos (condición de peligro **(NM, NC, B)**)
  - **$(nm\_oi < nc\_oi \wedge nm\_oi \neq 0) \vee (nm\_oi > nc\_oi \wedge nm\_oi \neq 3)$**
- Operadores: **cruzaM, cruzaMM, cruzaC, cruzaCC, cruzaMC**
  - Ejemplo de especificación para **cruzaM (NM, NC, B)**
    - Precondiciones
      - **$(b = 1 \wedge nm\_oi > 0) \vee (b = 0 \wedge nm\_oi < 3)$**
    - Postcondiciones
      - **$(nm\_oi, nc\_oi, 1) \rightarrow (nm\_oi - 1, nc\_oi, 0)$**
      - **$(nm\_oi, nc\_oi, 0) \rightarrow (nm\_oi + 1, nc\_oi, 1)$**
      - **Estado destino es un estado permitido**
    - Coste del operador: 1
  - Especificación para **cruzaMM, cruzaC,**...
- Coste de la solución = número de operadores aplicados

Representación: pensar en estados y operadores

- La representación de los estados afecta a
  - la facilidad/dificultad de la especificación de operadores
  - y a la complejidad para resolver el problema
- Ejemplo de los misioneros
  1. Formalización: *(m1, m2, m3, c1, c2, c3, B)*
     - *cruzaM1 (m1, m2, m3, c1, c2, c3, b)*
       - Precondiciones: *{ m1 = b }*
       - Postcondiciones:

         si b = izquierda entonces m1 := derecha $\wedge$ *b := derecha*

         si B = derecha entonces M1 := izquierda $\wedge$ *b := izquierda*

         estado destino no peligroso
     - Habría que especificar 21 operadores
       - 3 para cruzar a un misionero, 3 para cruzar a un caníbal
       - 3 para cruzar a dos misioneros, 3 para cruzar a dos caníbales
       - 9 para cruzar a un caníbal y a un misionero
  2. Formalización: *(nm_oi, nc_oi, nm_od, nc_od, b)*
     - La información redundante supone hacer cambios en más componentes

Representación: pensar en estados y operadores

- Ejemplo del 8-puzzle: posibilidades para los operadores
  - Centrarse en los estados era implanteable (1 operador / 1 estado)
    - 9!*4 = 1.451.520 operadores
  - Centrarse en la ficha a mover era factible
    - 8*4 operadores
    - Puede parametrizarse la especificación de los operadores
      - Dependiendo de cómo se haga la representación de estados
      - Debe facilitar localizar y cambiar cuál es la posición de una ficha concreta
      - *izquierda(Ficha), derecha(Ficha), arriba(Ficha), abajo(Ficha)*
    - Aunque así sea más fácil, seguirían saliendo 32 operadores
  - Centrarse en el hueco (la mejor opción)
    - Salían 4 operadores

Tema 2: Resolución de problemas mediante búsqueda

1. Representación de problemas en un espacio de estados
2. Resolución de problemas con estrategias de búsqueda
   - Métodos no informados o ciegos
   - Métodos informados o heurísticos
   - Búsqueda entre adversarios

Búsqueda ciega/ no informada

Cuando no tenemos información adicional acerca de los estados más allá de la definición del problema. Sólo podemos generar estados sucesor y comprobar si es la solución o no.

- Exploración exhaustiva del espacio de búsqueda
  - hasta encontrar una solución
- No incorporan conocimiento que guíe la búsqueda
  - Se decide a priori qué camino sigue

    p.e.: primero en profundidad
- La búsqueda no incorpora información del dominio

Estrategias de búsqueda

- **Búsqueda no informada (ciega):**
  - Las distintas estrategias de búsqueda difieren en el **orden** en el que los nodos se van expandiendo.
    - Búsqueda primero en anchura.
    - Búsqueda de coste uniforme.
    - Búsqueda primero-en-profundidad.
    - Búsqueda de profundidad limitada.
    - Búsqueda primero en profundidad con profundidad iterativa
    - Búsqueda bidireccional.

- **Búsqueda informada (heurística)**
  - Incorpora conocimiento del dominio para guiar la búsqueda en forma de "pistas" para guuar el proceso de búsqueda y hacerlo más eficiente
  - **Función heurística:** h(n)

    El valor de la heurística para el estado n proporciona una estimación del coste de la trayectoria óptima a partir el estado n hasta un estado objetivo.

Árbol de búsqueda, I

- En casi todos los casos, la búsqueda es en un **grafo de búsqueda** (puede haber múltiples caminos desde el nodo inicial a un nodo dado).
- Enfoquémonos en búsqueda en un **árbol** (un sólo camino desde el nodo raíz a un nodo dado)
  - Los **nodos de un árbol de búsqueda** corresponden a estados de búsqueda.
  - El **nodo raíz** de un árbol de búsqueda corresponde al estado de búsqueda inicial.
  - **Acciones**: **Expandir** el nodo de búsqueda actual: Generar nodos hijo (correspondientes a nuevos estados) aplicando la función **sucesor** al nodo actual.
  - **Estado objetivo:** Un nodo correspondiente a un estado que satisface el **test de objetivo**.
  - **Utilidad**: Coste del camino desde el nodo raíz al nodo actual.

Terminología:

- **Nodo padre**: Nodo del árbol desde el cual se ha generado el nodo actual aplicando una sola vez la función sucesor.
- **Nodo ancestro**: Un nodo en el árbol desde el cual el nodo actual ha sido generado aplicando una o varias veces la función sucesor.
- **Profundidad**: Longitud del camino desde la raíz al nodo actual.
- **Nodo hoja**: Nodo generado que no se ha expandido aún
- **Frontera**: Conjunto formado por los nodos hoja.

Búsqueda en árbol: sin eliminación de estados repetidos

- Pseudocódigo para búsqueda en árbol

  ***problema* = {*nodo-raíz*, *expandir*, *test-objetivo*}**

  estrategia

function búsqueda-en-árbol (*problema*, e*strategia*)

;; devuelve *solución* o *fallo*

;; *lista-abierta* contiene los nodos de la frontera del árbol de búsqueda

Inicializar *árbol-de-búsqueda* con *nodo-raíz*

Inicializar *lista-abiertos* con *nodo-raíz*

Iterar

If (*lista-abiertos* está vacía) **then return** *fallo*

Elegir de *lista-abiertos*, de acuerdo a la *estrategia*, un *nodo* a expandir.

If (*nodo* satisface *test-objetivo*)

then return *solución* (camino desde el *nodo-raíz* hasta el *nodo actual*)

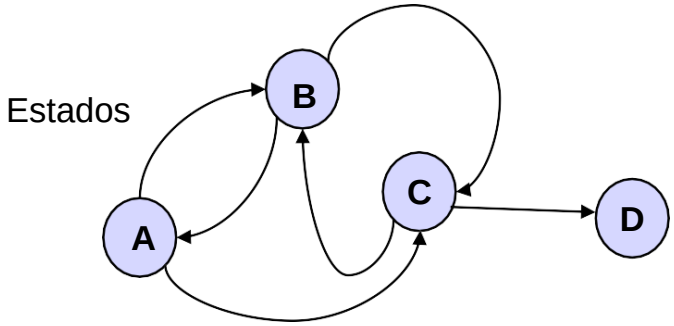
else eliminar *nodo* de *lista-abiertos*

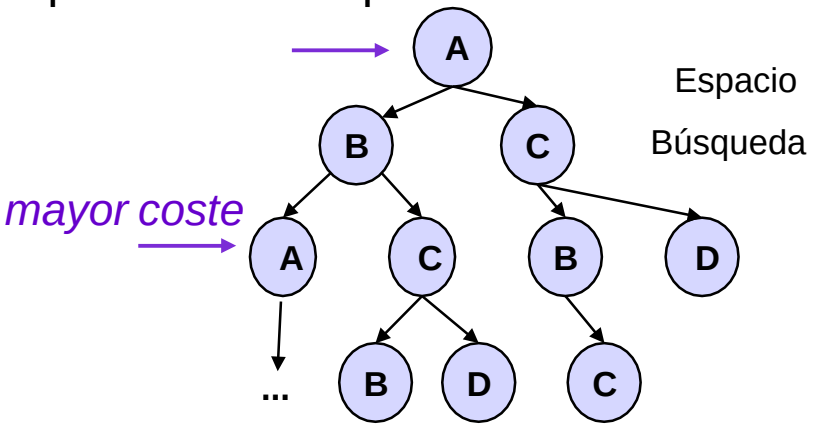
expandir nodo

añadir nodos hijo a *lista-abiertos*

Espacio de "búsqueda" o árbol de búsqueda

- Árbol sólo con los nodos generados en la búsqueda de la solución
  - Depende del algoritmo de búsqueda utilizado (la estrategia)
  - Aunque el espacio de estados sea finito,…
  - …el espacio de búsqueda puede ser infinito
  - Nodos distintos del árbol de búsqueda pueden corresponder al mismo estado en el espacio de estados





- Un nodo en el árbol representa un camino desde la raíz hasta un cierto estado

Tipo de datos NODO del *árbol de búsqueda*:

(estado, nodo_padre, operador, profundidad, coste_camino_acumulado)

Recordamos Terminología

- Estructura lista-abiertos

  Se guardan nodos con estados generados pendientes de expandir

- Estructura lista-cerrados

  Se guardan los ya expandidos o visitados (solo en algunos algoritmos)

- Tipos de nodos
  - **Generados**

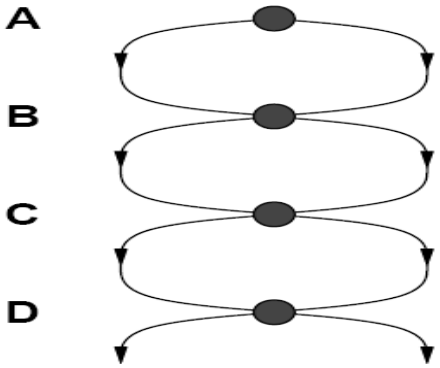
    Son aquéllos que aparecen o han aparecido en nodos de *lista-abiertos*
  - **Generados pero no expandidos**

    Son aquéllos que aparecen en *lista-abiertos*
  - **Expandidos:** Son aquéllos que aparecen en *lista-cerrados*
    - Un estado se expande cuando
      1. Un nodo que lo representa se quita de *lista-abiertos*,
      2. Se generan todos sus descendientes y éstos se añaden a *lista-abiertos*
      3. Se incorpora a *lista-cerrados*,
    - Un estado puede ser expandido varias veces (en diferentes nodos)
      - Un nodo se expande una única vez

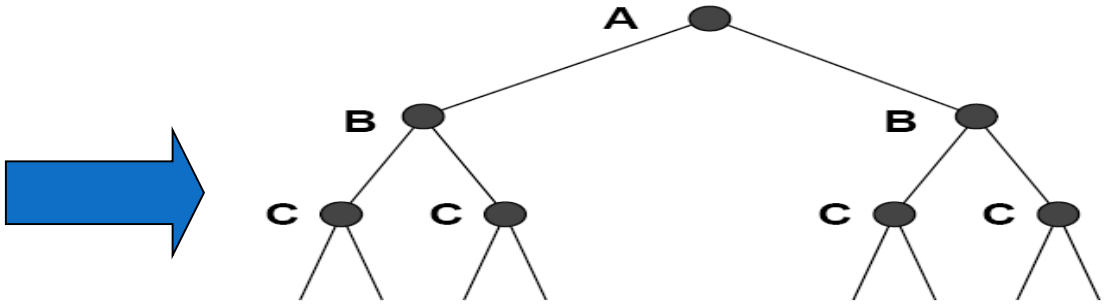
Eliminación de estados repetidos

- Si la repetición de estados no se detecta, la **complejidad** del problema de búsqueda puede incrementarse de manera **exponencial**
  - Por ejemplo, en una búsqueda en una malla regular
  - Ejemplo trivial

Asumiendo una profundidad máxima **m**



El árbol de búsqueda tiene 2^m nodos hoja



- **Eliminación de estados repetidos:**

  Guardar los nodos expandidos en *lista-cerrados*.
  - No expandir un nodo candidato si ya está en *lista-cerrados*
    - Requerimientos extra de memoria.
    - Óptimo en búsqueda con coste uniforme, y en búsqueda primero en anchura cuando el coste de todas las acciones es idéntico.
    - No se garantiza encontrar el camino óptimo con otras estrategias.

Búsqueda en grafo: con eliminación de estados repetidos

- Pseudocódigo para búsqueda en grafo

problema* = {*nodo-raíz*, *expandir*, *test-objetivo*}; *estrategia

function búsqueda-en-grafo (*problema*, *estrategia*)

;; devuelve *solución* o *fallo*

;; *lista-abiertos* contiene los nodos de la frontera de *árbol-de-búsqueda*

Inicializar *árbol-de-búsqueda* con *nodo-raíz*

Inicializar *lista-abiertos* con *nodo-raíz*

Inicializar *lista-cerrados* a una lista vacía

Iterar

If (*lista-abiertos* está vacía) **then return** *fallo*

Elegir dentro de *lista-abiertos*, de acuerdo a la *estrategia*, un *nodo* a expandir.

If (*nodo* satisface *test-objetivo*)

then return *solución* (camino desde *nodo-raíz* a *nodo*)

eliminar *nodo* de *lista-abiertos*

If (*nodo* no está en *lista-cerrados*)

then añadir *nodo* a *lista-cerrados*

expandir *nodo*

añadir nodos hijo a *lista-abiertos*

Evaluación de las estrategias de búsqueda

Rendimiento de un algoritmo en la resolución de problemas de búsqueda

- Completitud. ¿garantiza encontrar solución si esta existe?
- Optimalidad. ¿encuentra la solución de coste mínimo?
- Coste de la solución: dado por la función de utilidad.
- Coste computacional de la búsqueda
  - Complejidad temporal. ¿cuánto tarda en encontrar una solución? Es el peor caso: el nodo objetivo el último del árbol (nº de nodos expandidos)
  - Complejidad espacial (uso de memoria). ¿cuánta memoria se necesita en el peor caso? (máximo nº de nodos que es necesario almacenar simultáneamente en memoria)

- Coste total: 2 tipo de costes
  - Coste de la solución: suma del coste operadores usados en el camino
    - Problemas de optimización: coste de la solución (independiente del camino)
  - Coste computacional de la búsqueda: complejidad del algoritmo
- Hay que alcanzar un compromiso entre ambos costes
  - Obtener la mejor solución posible con los recursos disponibles
    - No quiero una solución buena pero que cueste (tiempo, memoria) encontrar

Evaluación de las estrategias de búsqueda

Coste de la búsqueda (análisis de complejidad):

- Factor de ramificación (b): número máximo de sucesores de cualquier nodo.
- Profundidad del nodo objetivo menos profundo (d)
- Profundidad máxima del árbol de búsqueda (m)
- Coste mínimo de una acción (ε)

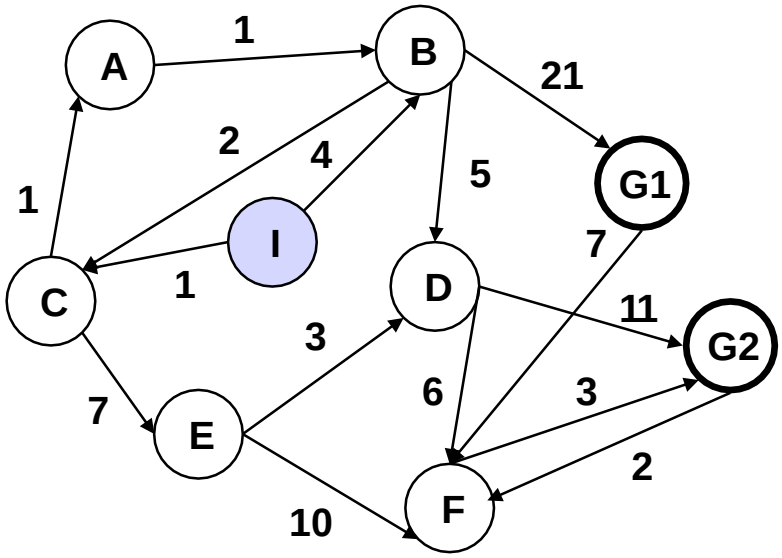
- Suponemos que:
  - El factor de ramificación (b) es finito.
  - Desde el punto de vista del análisis de eficiencia computacional, todas las operaciones tienen el mismo coste
  - La profundidad máxima del árbol de búsqueda (m) puede ser infinita.
  - Los costes son aditivos: Coste del camino = suma de costes de cada paso.

Búsqueda primero en anchura

- Los nodos se expanden por orden de *profundidad*
  - Nodos de profundidad *p* se expanden antes que los nodos de profundidad *p+1*
  - Usar para la *lista-abiertos* una cola FIFO (*first-in-first-out*) para la lista de candidatos a expandir.
    Los nodos nuevos generados se insertan al final de la cola ⇒ los nodos más someros se expanden antes que los más profundos.
- Propiedades (criterios de evaluación)
  - Completa: garantiza encontrar una solución (la menos profunda de entre las posibles)
  - Óptima: solo si el coste del camino es una función no decreciente de la profundidad del nodo (el camino de menor longitud puede no ser óptimo)
  - La complejidad temporal es exponencial
    - Suponiendo un factor de ramificación *máximo b* (nº de hijos de un nodo) y un camino hasta la solución de menor profundidad *d*,
    - el nº de nodos expandidos en el caso peor es $1+b+b^2+...+b^d-1=\frac{b^{d+1}-b}{b-1}=O(b^d)$
    - el nº de nodos generados en el caso peor es $b+b^2+...+b^d+\left(b^{d+1}-b\right)=\frac{b^{d+2}-b^2}{b-1}=O(b^{d+1})$
  - La complejidad espacial es exponencial $O(b^d)$
    - Todos los nodos generados han de mantenerse en memoria
    - Problema mayor: la memoria (sólo viable para casos pequeños)
- Ciclos: Con *busqueda-en-arbol* (sin eliminación de estados repetidos) no se pierden soluciones, aunque suponen ineficiencia.
- Si no hay solución, el algoritmo podría no terminar.

Ejemplo: búsqueda primero en anchura

Espacio de estados

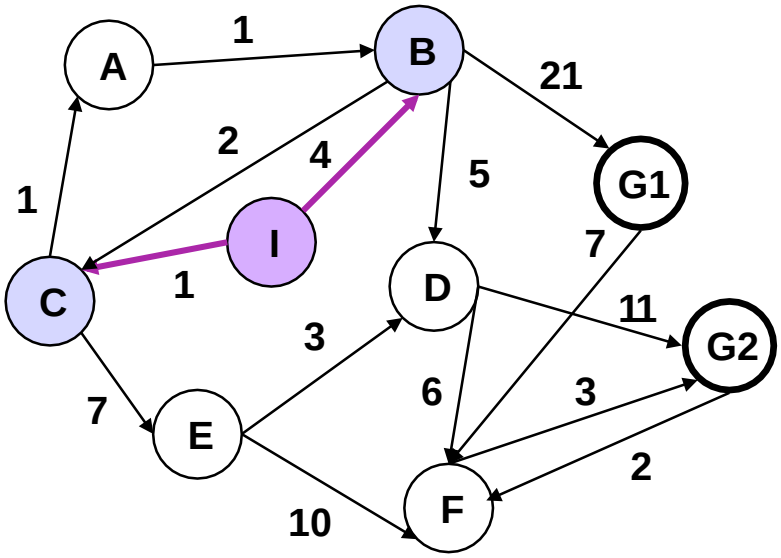


Árbol de búsqueda

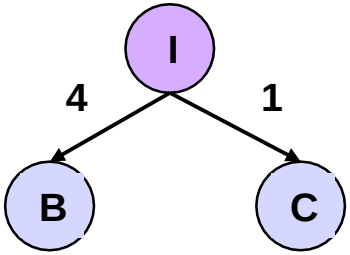
lista-abiertos (cola): (I)

Ejemplo: búsqueda primero en anchura

Espacio de estados



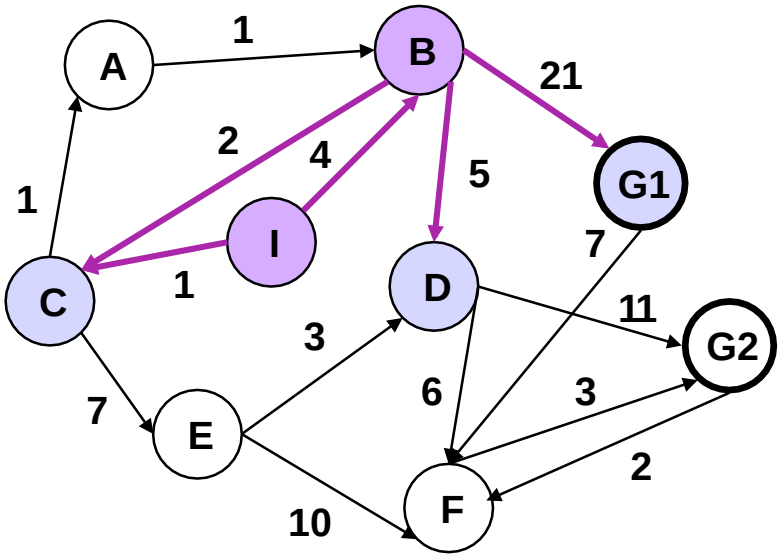
Árbol de búsqueda



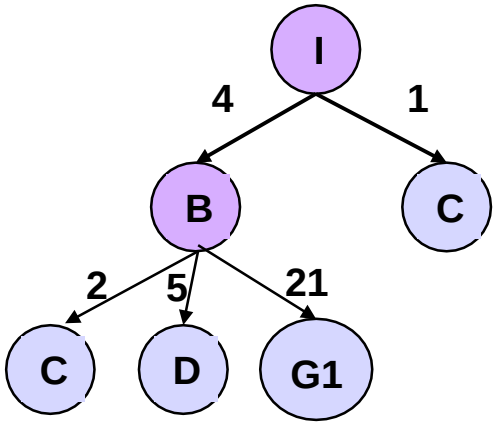
lista-abiertos (cola): (B C)

Ejemplo: búsqueda primero en anchura

Espacio de estados



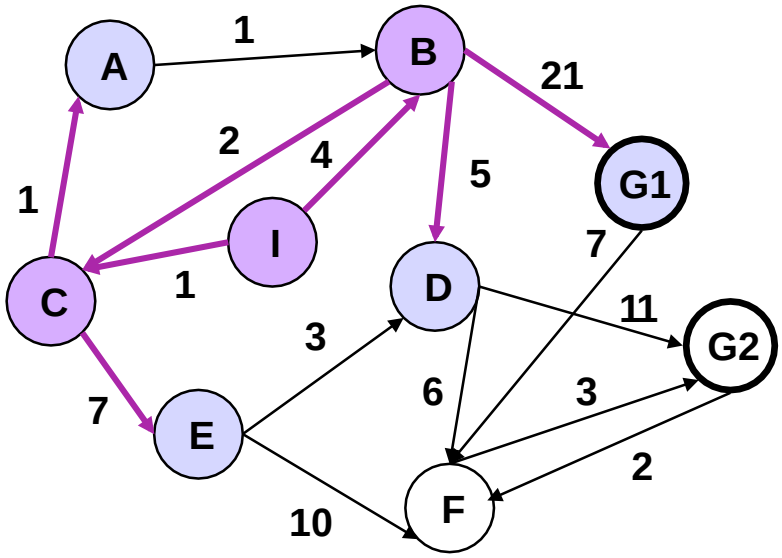
Árbol de búsqueda



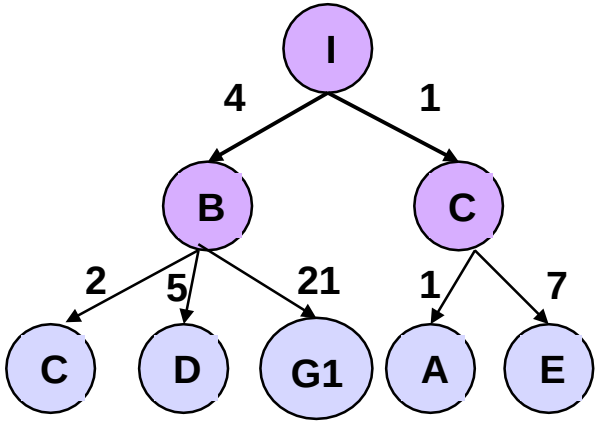
lista-abiertos (cola): (C C D G1)

Ejemplo: búsqueda primero en anchura

Espacio de estados



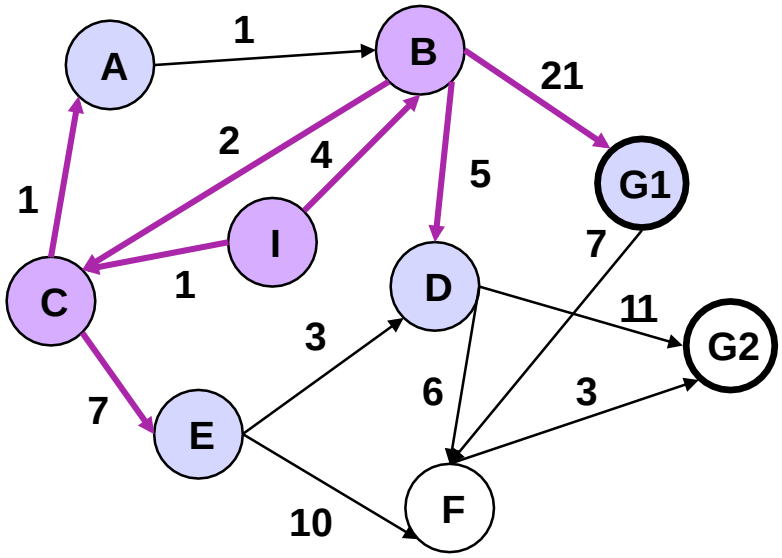
Árbol de búsqueda



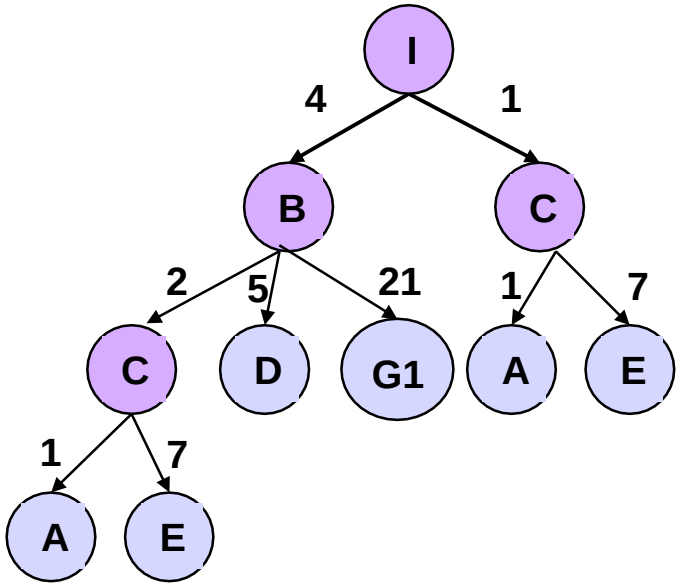
lista-abiertos (cola): (C D G1 A E)

Ejemplo: búsqueda primero en anchura

Espacio de estados



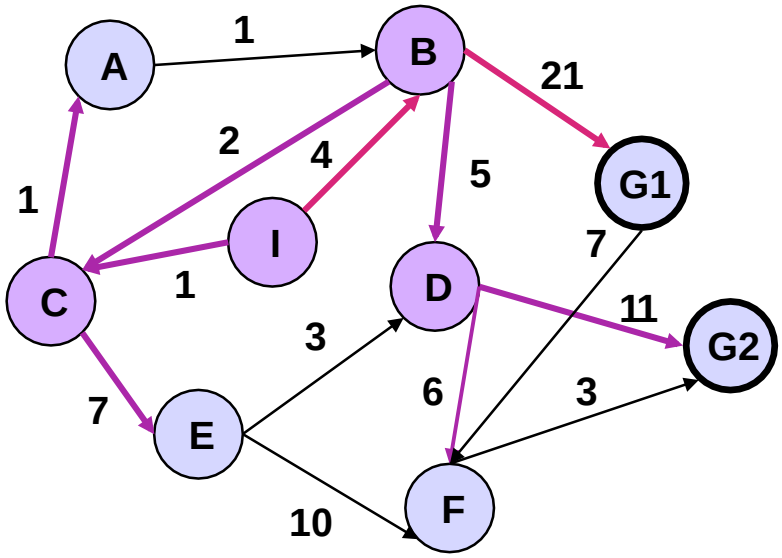
Árbol de búsqueda



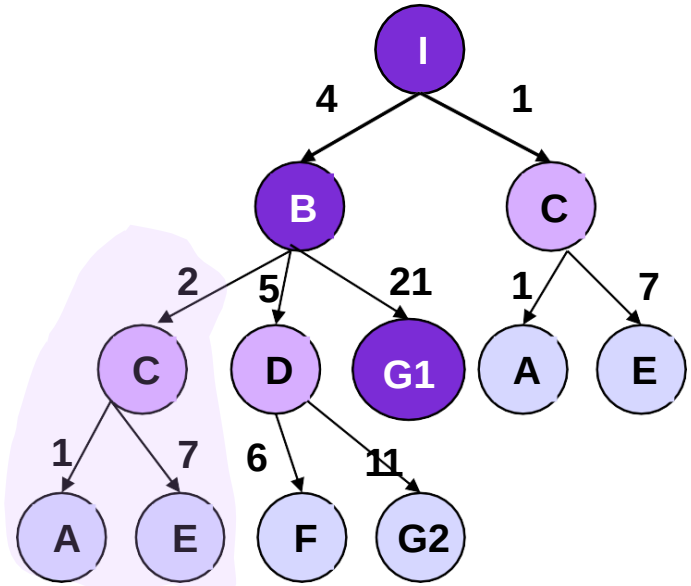
lista-abiertos (cola): (D G1 A E A E)

Ejemplo: búsqueda primero en anchura

Espacio de estados



Árbol de búsqueda



Nodos expandidos (por orden): I B C C D G1

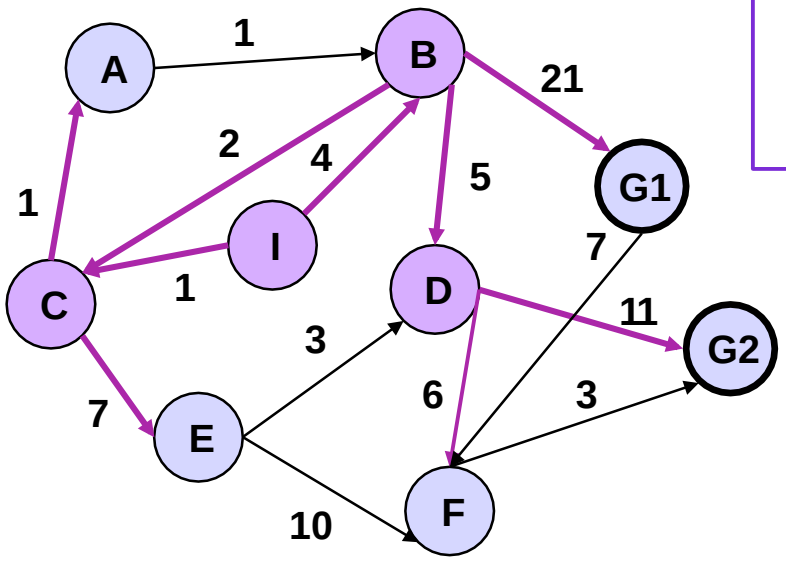
Nodos generados (por orden): I B C C D G1 A E A E F G2

Camino a la solución: I B G1 Coste: 4+21 = 25

IMPORTANTE. Hasta que no se visita el nodo no se encuentra la solución. Aunque esté ya generado

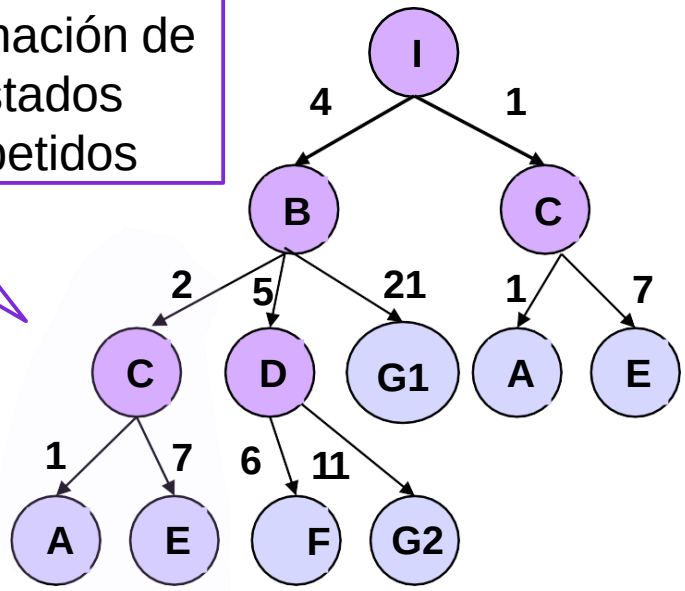
Ejemplo: búsqueda primero en anchura

Espacio de estados



Árbol de búsqueda

Eliminación de estados repetidos

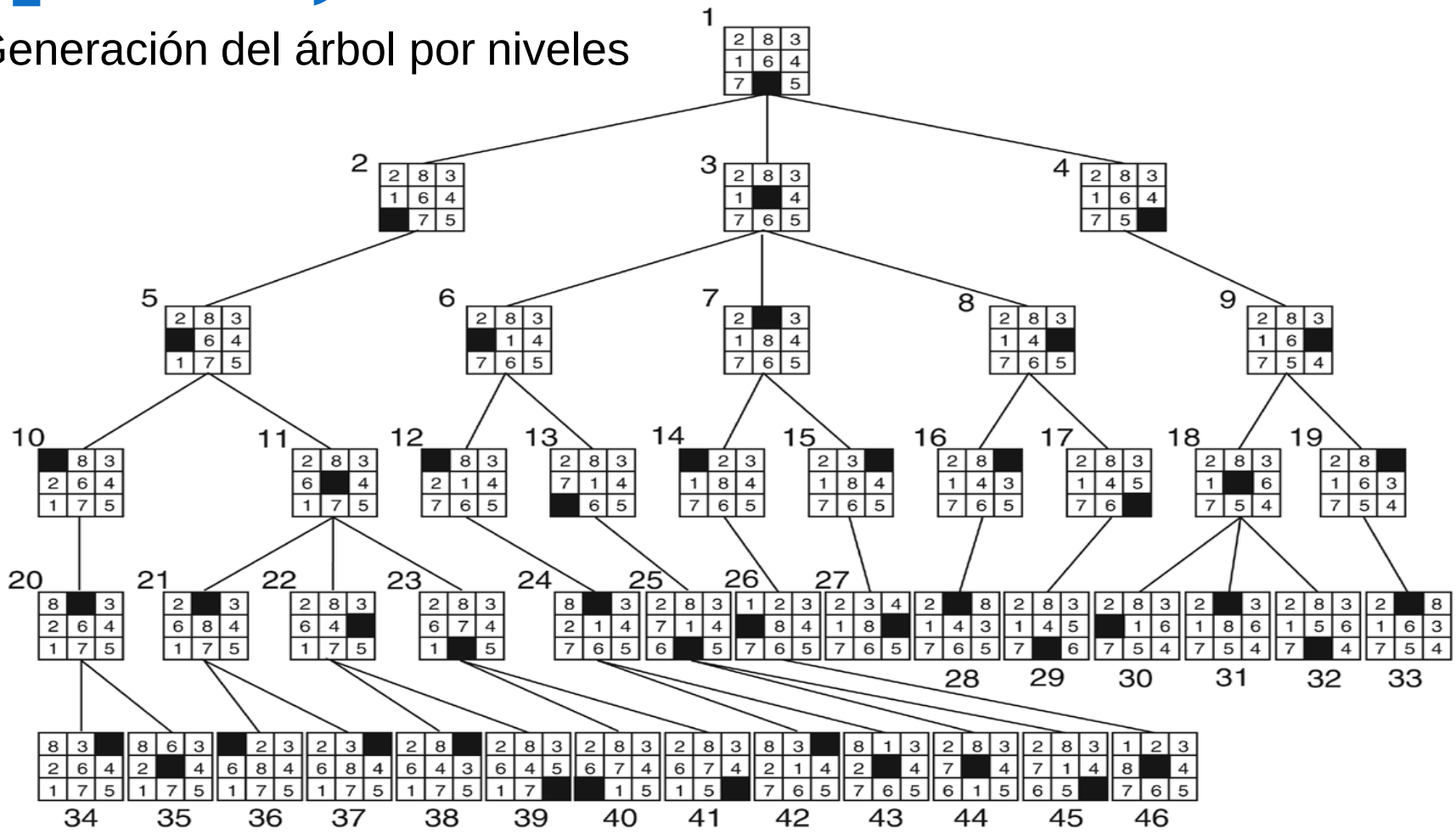


lista-abiertos (cola): (G2 F E A E A G1)

- Es completa: Encuentra la solución menos profunda.
- No es óptima: La solución encontrada no es la de menor coste, ya que porque el coste de camino no se incrementa de manera monótona con la profundidad.

Búsqueda primero en anchura (8-puzzle)

- Generación del árbol por niveles



Búsqueda de coste uniforme

- Los nodos se expanden por orden creciente de *coste* del camino (*acumulado)*
  - En cada paso: De los nodos en *lista-abiertos*, se expande el nodo que tiene el menor coste del camino hasta llegar a él

- *lista-abiertos* se implementa con una cola de prioridad (nodos ordenados de menor a mayor coste de camino)
- Coste del camino frente al número de pasos

  Si el coste del camino de los nodos se incrementa de manera monótona con su profundidad → búsqueda de coste uniforme equivale a la búsqueda primero en anchura
- Propiedades:
  - Completa si no existen caminos de longitud infinita y coste finito.
  - Óptima si *coste-camino(sucesor(n))* $\geq$ *coste-camino(n)*
    - Se satisface cuando todos los operadores tienen *coste* ≥ 0
  - Complejidad en espacio y tiempo equivalente a primero en anchura si el coste del camino de los nodos se incrementa de manera monótona con su profundidad: $O(b^d)$
  - *Si no, en el caso peor:*

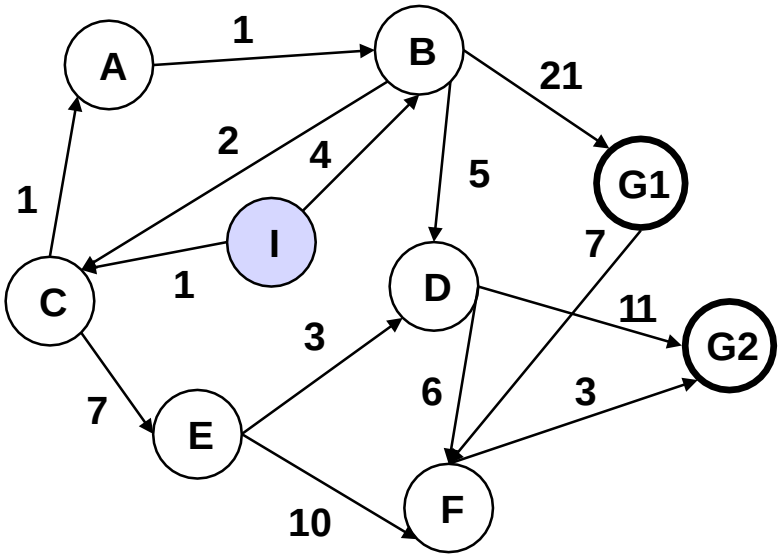
$$O\left(b^{\lceil C^*/\varepsilon \rceil}\right)$$

$C^* \equiv$ Coste de camino de la solución óptima

$\varepsilon \equiv$ Coste mínimo (>0) de una acción

Ejemplo: búsqueda de coste uniforme

Espacio de estados

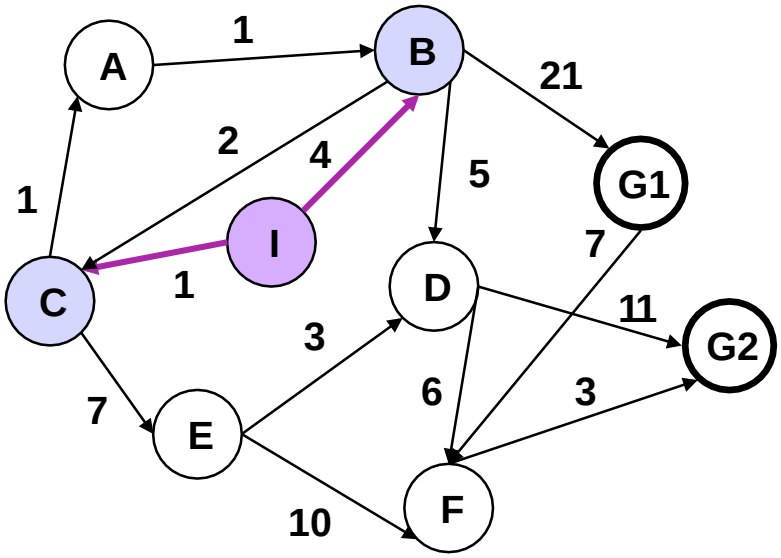


Árbol de búsqueda

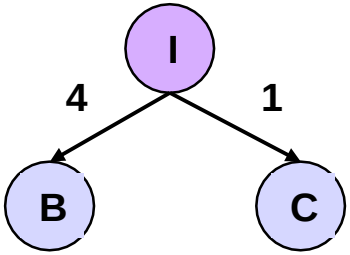
lista-abiertos (cola de prioridad): (I_0)

Ejemplo: búsqueda de coste uniforme

Espacio de estados



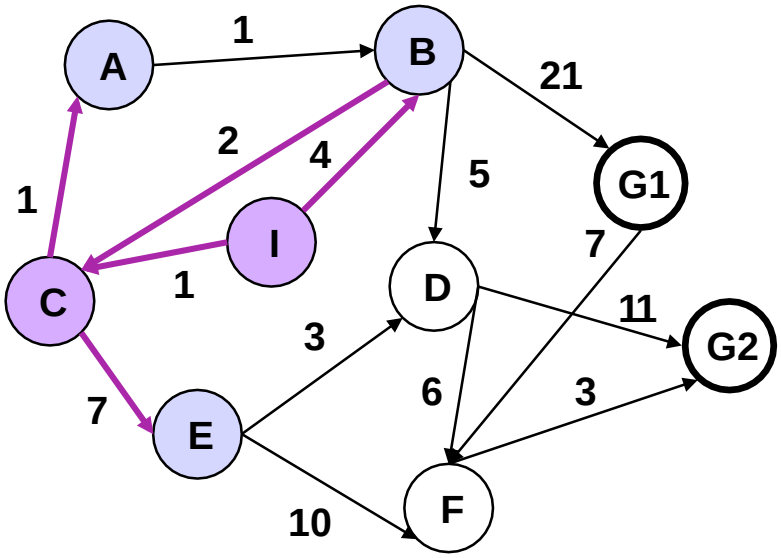
Árbol de búsqueda



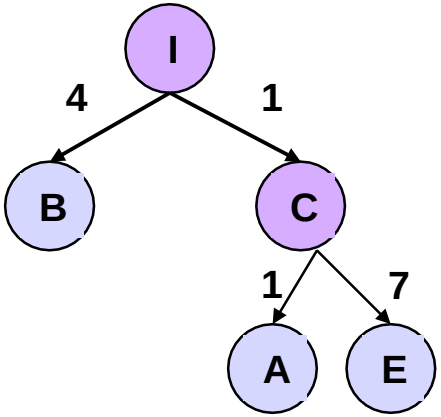
lista-abiertos (cola de prioridad): $(C_1\ B_4)$

Ejemplo: búsqueda de coste uniforme

Espacio de estados



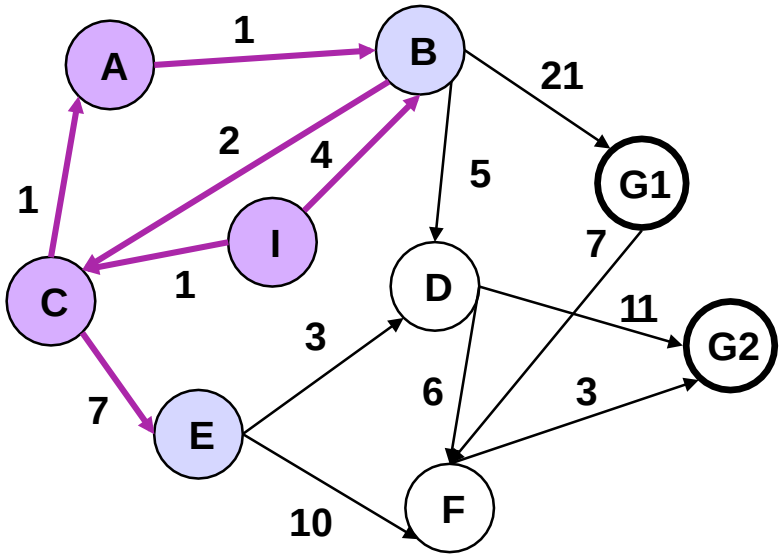
Árbol de búsqueda



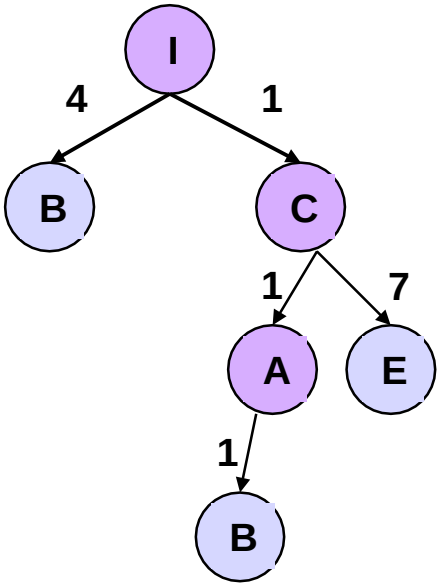
lista-abiertos (cola de prioridad): (A_2 B_4 E_8)

Ejemplo: búsqueda de coste uniforme

Espacio de estados



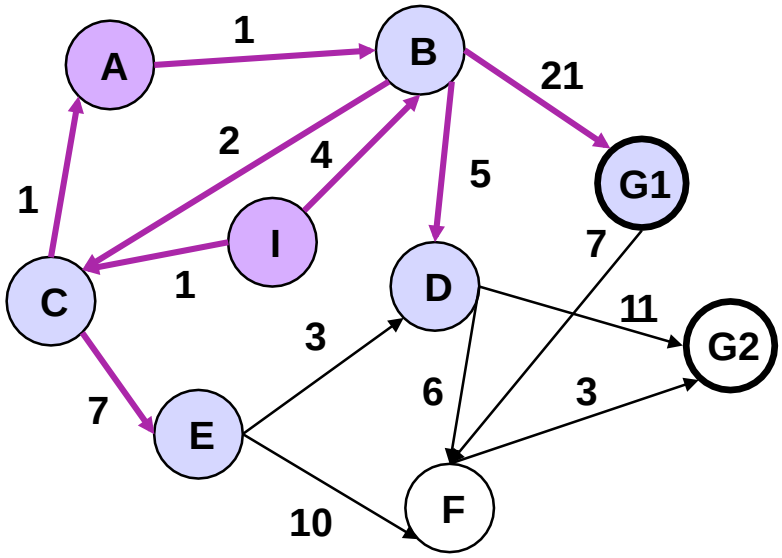
Árbol de búsqueda



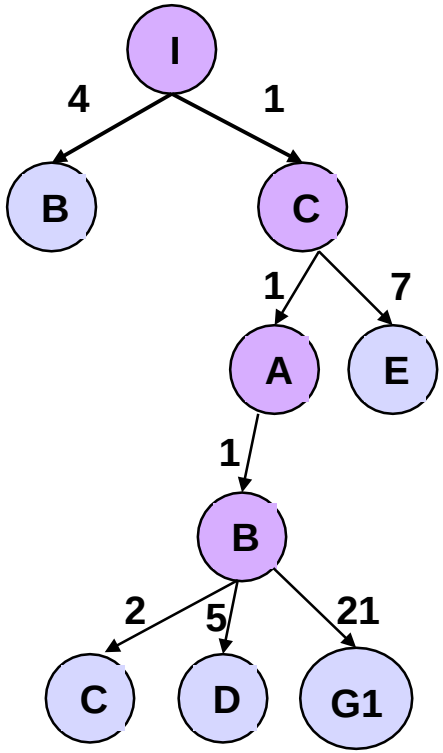
lista-abiertos (cola de prioridad): (B_3 B_4 E_8)

Ejemplo: búsqueda de coste uniforme

Espacio de estados



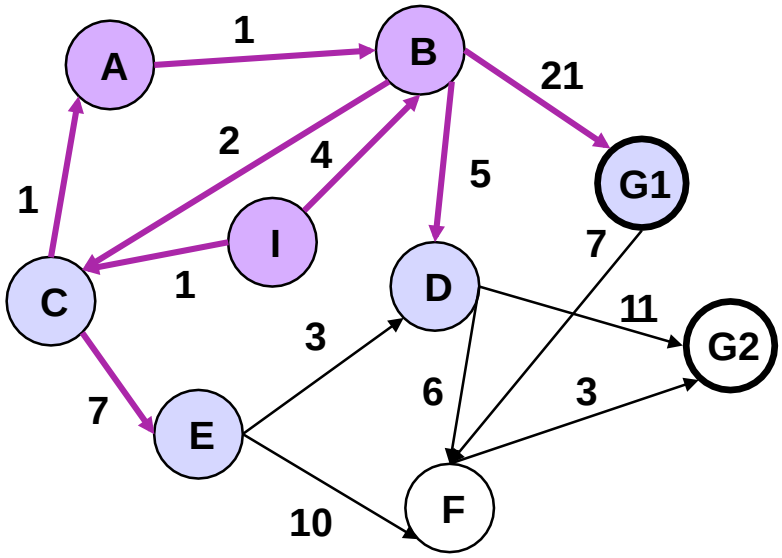
Árbol de búsqueda



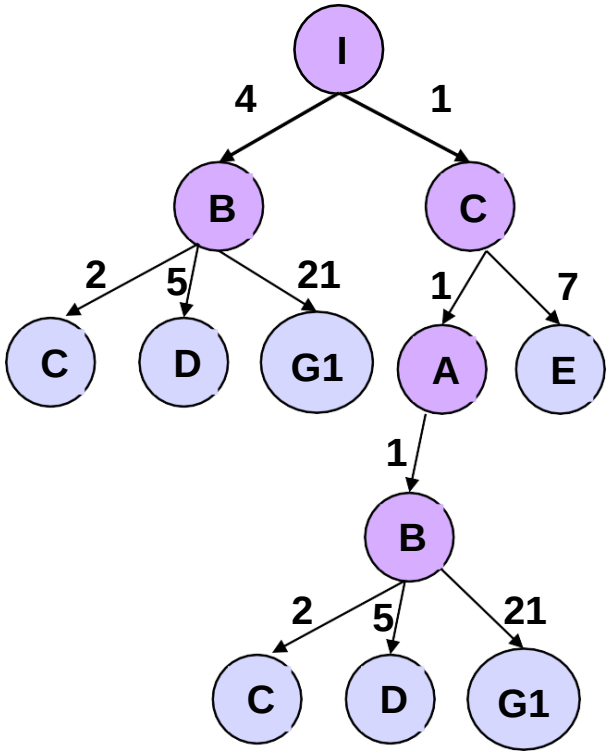
lista-abiertos (cola de prioridad): (B_4 C_5 D_8 E_8 $G1_{24}$)

Ejemplo: búsqueda de coste uniforme

Espacio de estados



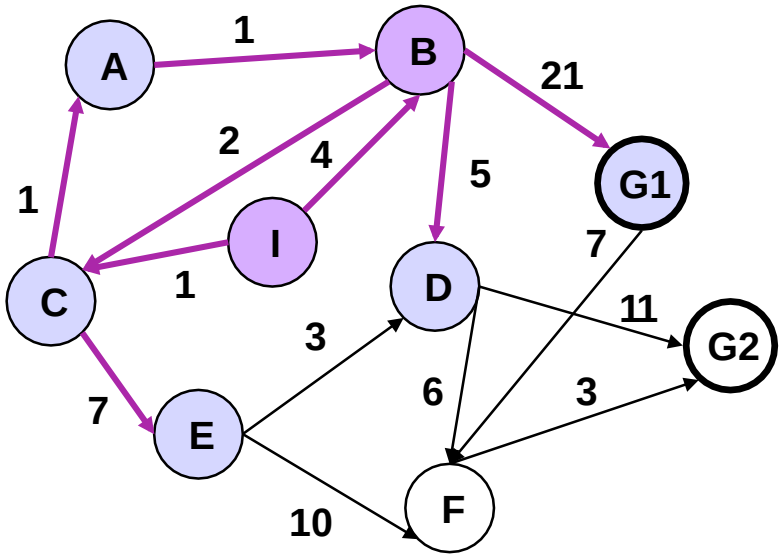
Árbol de búsqueda



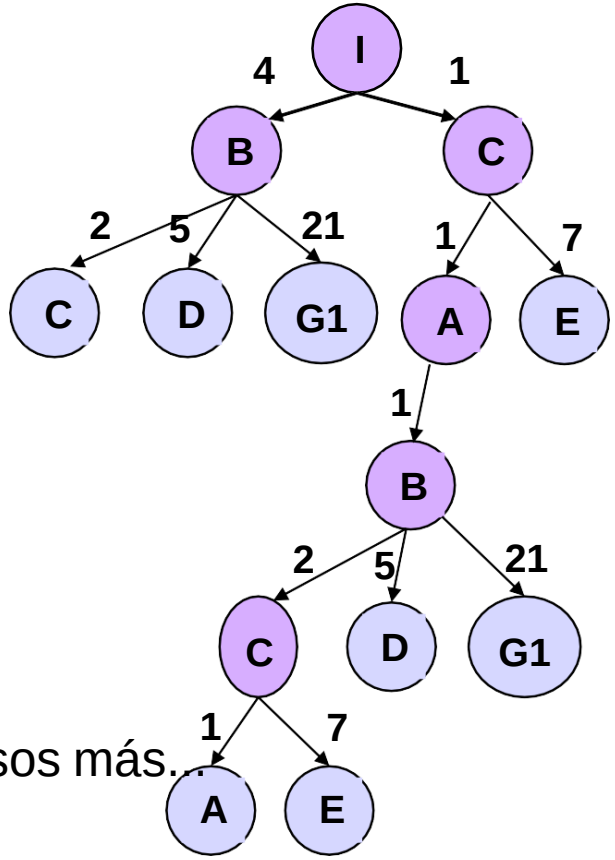
lista-abiertos (cola de prioridad): (C_5 C_6 D_8 E_8 D_9 $G1_{24}$ $G1_{25}$)

Ejemplo: búsqueda de coste uniforme

Espacio de estados



Árbol de búsqueda

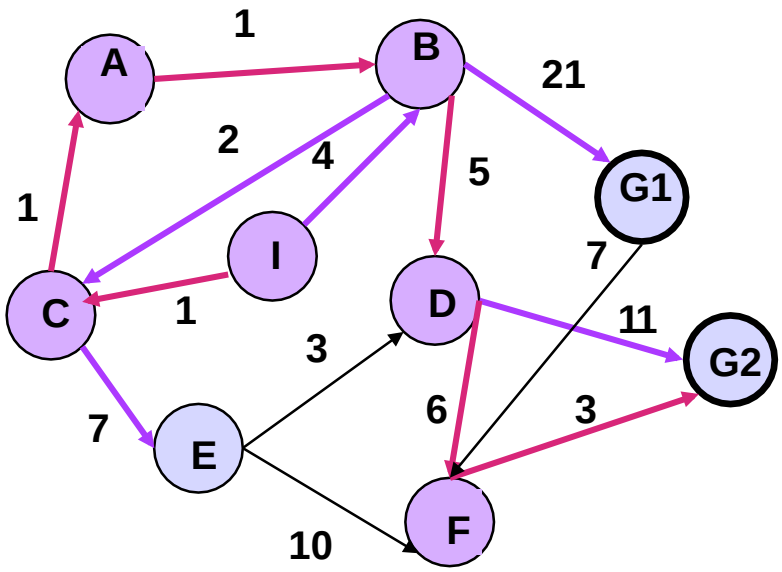


Y así seguiría unos cuantos pasos más...

lista-abiertos (cola de prioridad): (A_6 C_6 D_8 E_8 D_9 E_{12} $G1_{24}$ $G1_{25}$)

Ejemplo: búsqueda de coste uniforme

Espacio de estados



Camino a la solución: I C A B D F G2

Coste: 1+1+1+5+6+3 = 17

Búsqueda de coste uniforme

- Completa y óptima
- Expande muchos nodos
- Aquí es mucho mayor el problema de los ciclos
  - Pero un control de ciclos sobre *abiertos* supondría que la estrategia dejaría de ser óptima...
  - Puede verse en este ejemplo *(el B hijo de A no se generaría y no se encontraría esta solución óptima)*
  - *Control solo sobre expandidos sí Funciona*
- La solución de menor coste se encuentra a profundidad 6

Búsqueda primero en profundidad

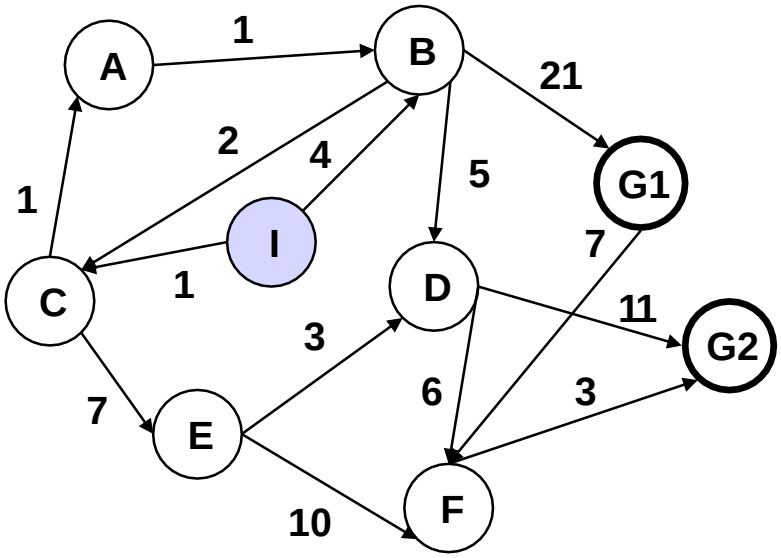
- Los nodos generados más recientemente son los primeros que se expanden
  - El nodo que se expande es el primero de *lista-abiertos.*
    - Se elimina de se elimina de *lista-abiertos.*
    - Si el nodo considerado no tiene sucesores y no es objetivo, se descarta

      En caso contrario se expande (y se elimina de *lista-abiertos*)
  - Implementación: Usar para la *lista-abiertos* una cola LIFO (*last-in-first-out*), (**pila**) para implementar la lista de candidatos a ser expandidos. Los nodos nuevos generados se introducen al principio de la cola => los nodos más profundos se expanden primero.
  - Alternativa: Función recursiva que se llama a sí misma por cada uno de sus hijos.
- La estructura *lista-abiertos* se implementa con una pila (o se usa recursión)
- Propiedades:
  - No completa: el algoritmo puede explorar caminos de longitud infinita.
  - No óptima: No hay garantía de que, en caso de encontrar una solución, esta sea la de menor coste. No recomendable si ***m*** (máxima profundidad del árbol) si este valor es grande.
  - Espacio: *O(b*m),*donde ***b*** es el factor de ramificación-

    Los requisitos modestos: basta almacenar de manera simultánea en memoria el camino actual y referencias a los hermanos de los nodos expandidos en esa rama.
  - Tiempo: $O(b^m)$ (si hay muchas soluciones, puede ser más rápida que primero en anchura; depende del orden de aplicación de los operadores).

Ejemplo: búsqueda primero en profundidad

Espacio de estados

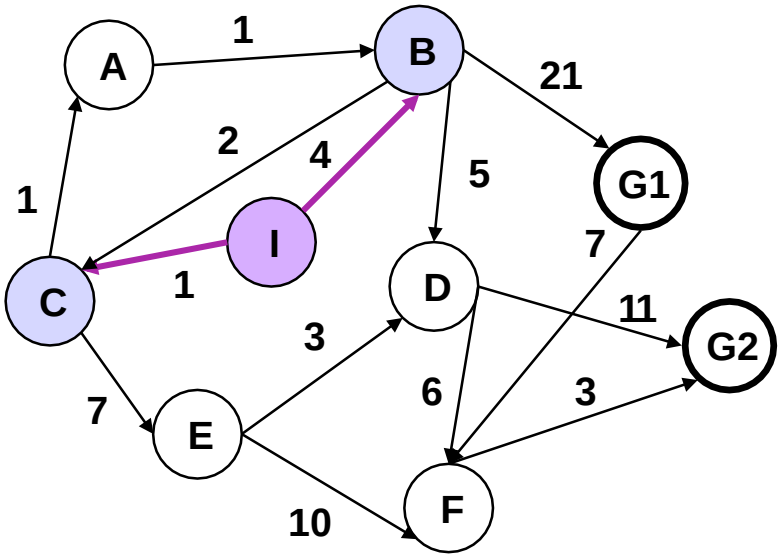


Árbol de búsqueda

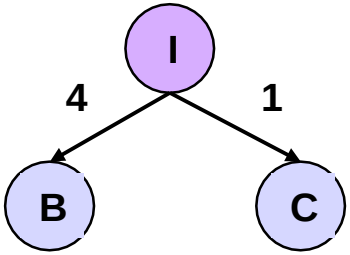
lista-abiertos (pila): (I)

Ejemplo: búsqueda primero en profundidad

Espacio de estados



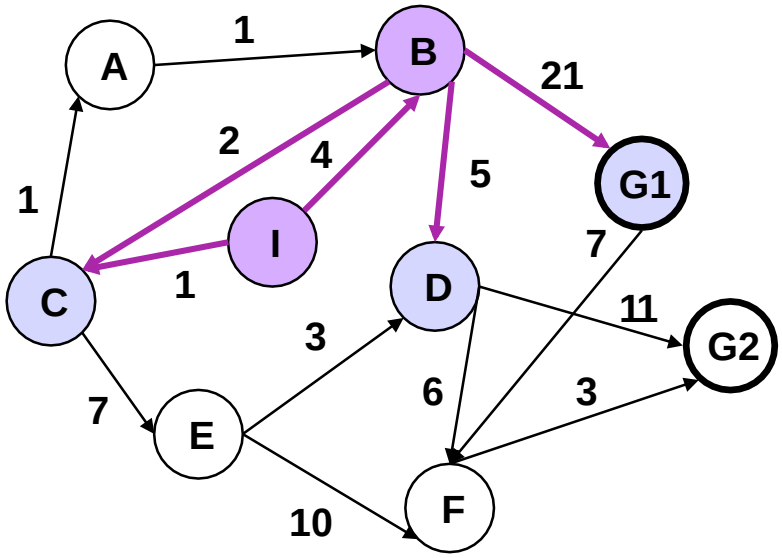
Árbol de búsqueda



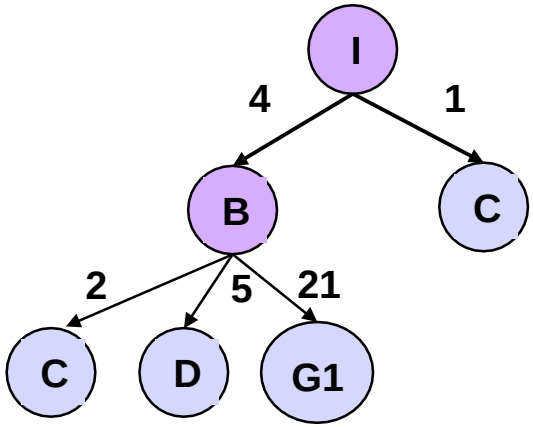
lista-abiertos (pila): (B C)

Ejemplo: búsqueda primero en profundidad

Espacio de estados



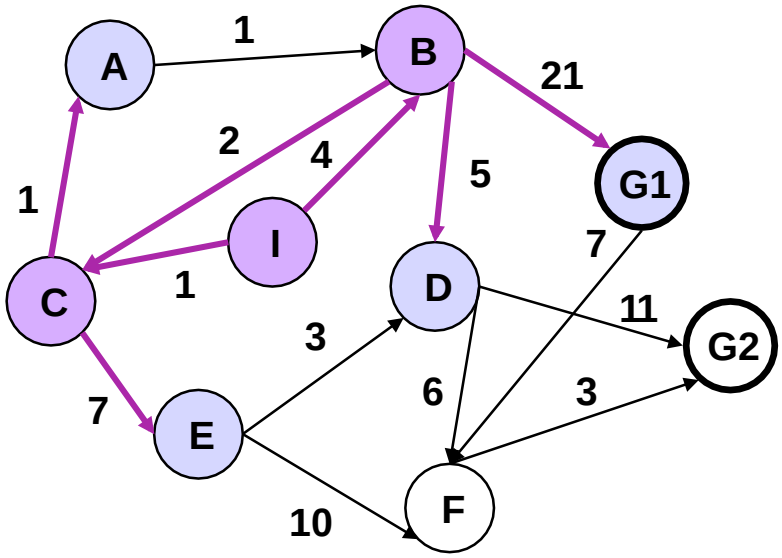
Árbol de búsqueda



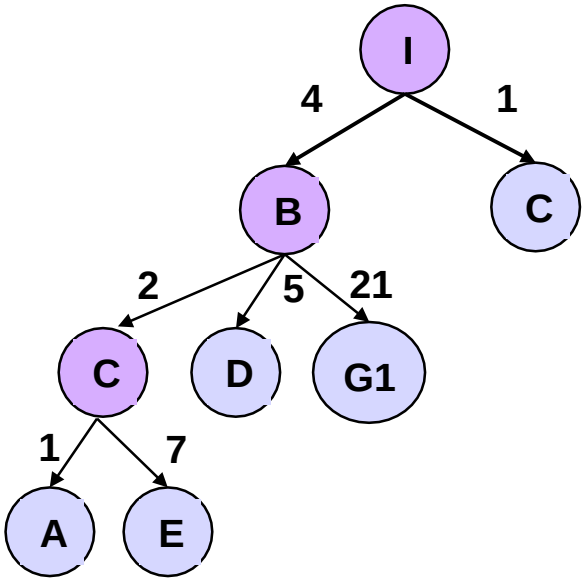
lista-abiertos (pila): (C D G1 C)

Ejemplo: búsqueda primero en profundidad

Espacio de estados



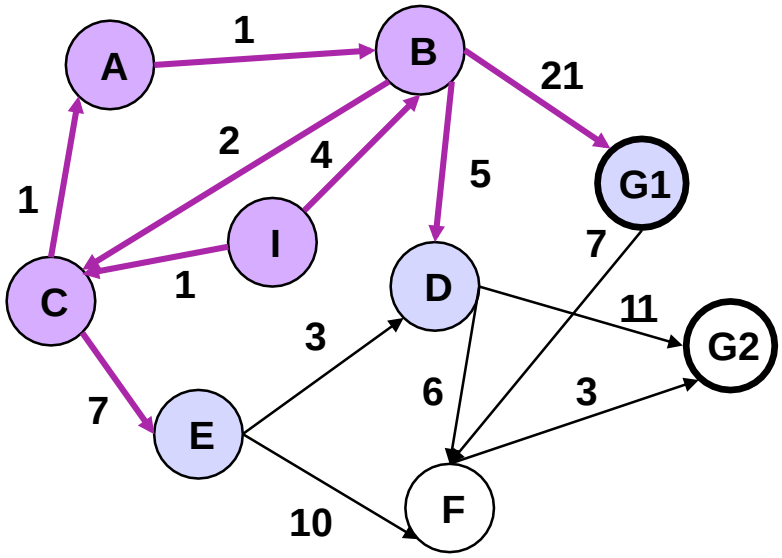
Árbol de búsqueda



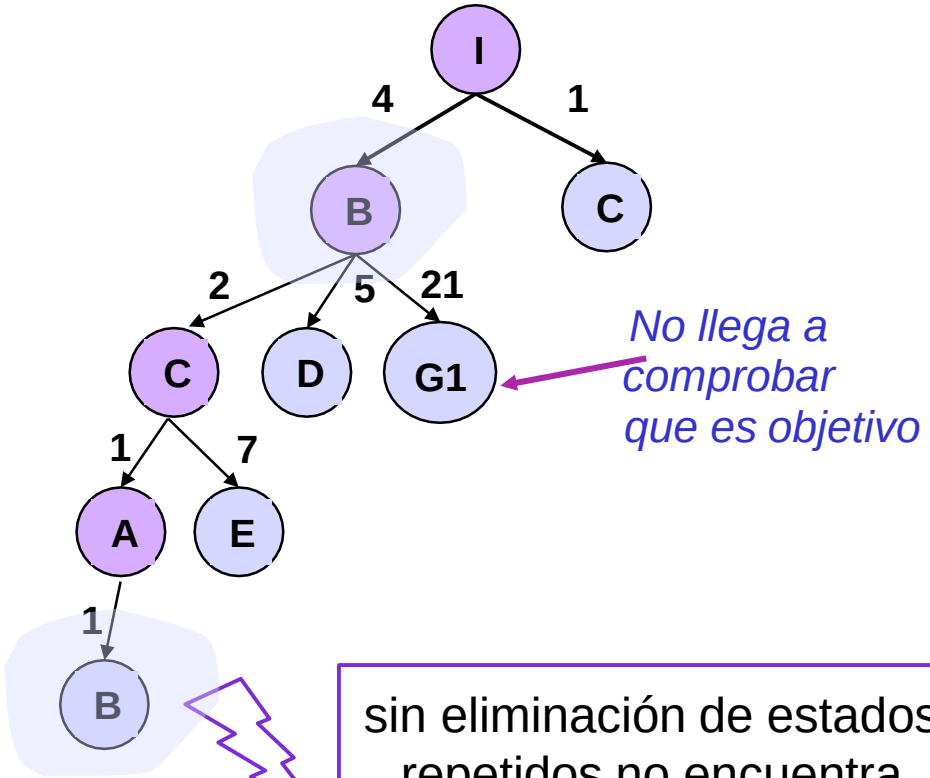
lista-abiertos (pila): (A E D G1 C)

Ejemplo: búsqueda primero en profundidad

Espacio de estados



Árbol de búsqueda



No llega a comprobar que es objetivo

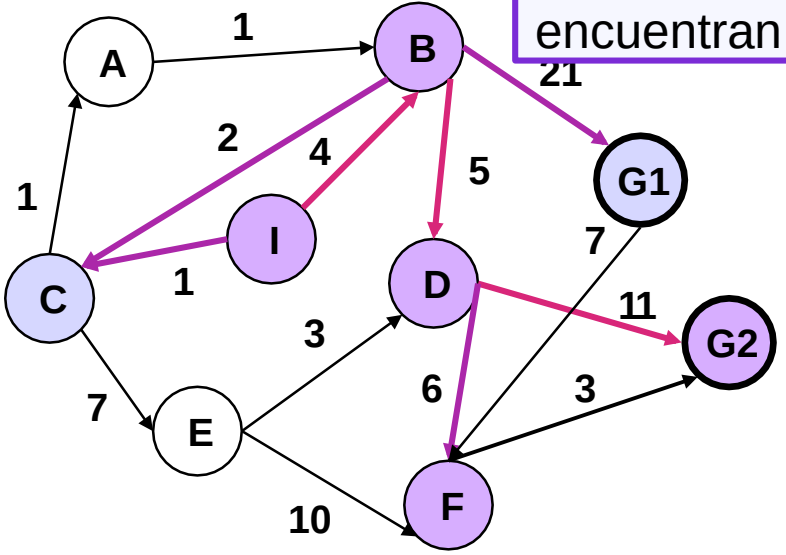
lista-abiertos (pila): (B E D G1 C)

Nodos expandidos: *lista-cerrados* (pila): (A C B I)

sin eliminación de estados repetidos no encuentra solución (explora una rama de longitud infinita)

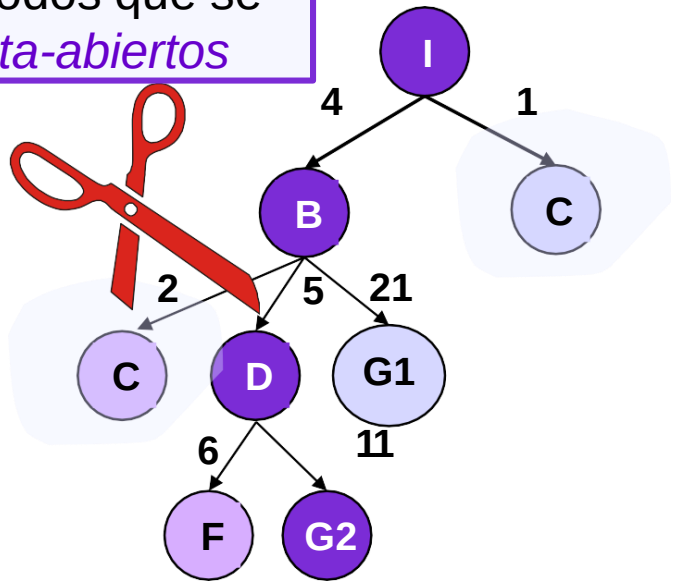
Primero en profundidad + eliminación de estados repetidos con *lista-abiertos*

Espacio de estados



Espacio de búsqueda

Evitar expandir nodos que se encuentran en *lista-abiertos*



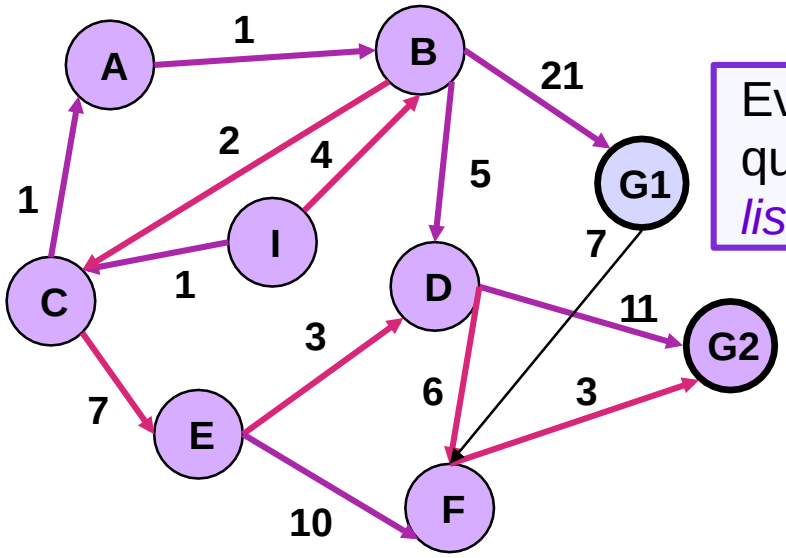
Nodos expandidos (por orden de expansión): I B D F G2

Nodos generados (por orden de generación): I B C D G1 F G2

Camino a la solución: I B D G2 Coste: 4+5+11 = 20

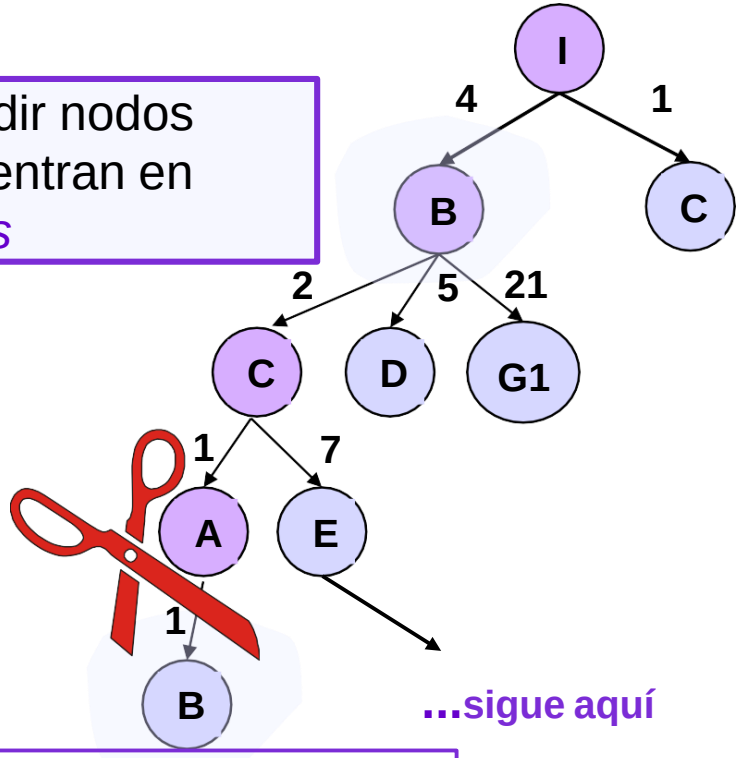
Primero en profundidad + eliminación de estados repetidos con *lista-cerrados* [preferible]

Espacio de estados



Evitar expandir nodos que se encuentran en *lista-cerrados*

Árbol de búsqueda



…sigue aquí

Nodos expandidos (por orden de expansión): I B C A E D F G2

Camino a la solución: I B C E D F G2 Coste: 4+2+7+3+6+3 = 25

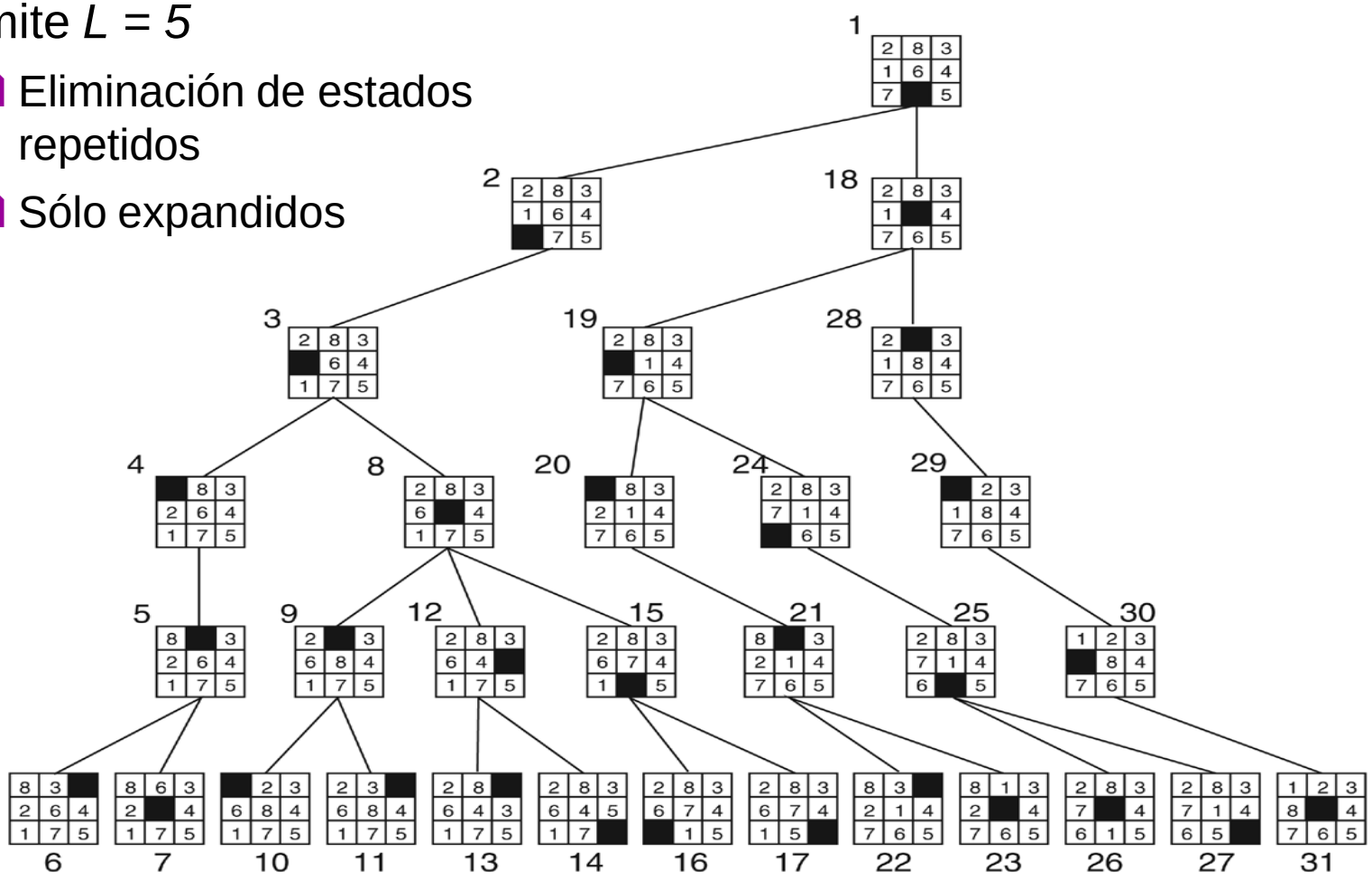
Búsqueda primero en profundidad con límite de profundidad

- Búsqueda en profundidad con límite de profundidad *L*
  - Implementación: Igual que la búsqueda *primero-en-profundidad*, suponiendo que los nodos con profundidad igual a *L* no tienen sucesores.
  - Se desechan caminos cuya profundidad es superior a *L.*
  - Evita descender indefinidamente en el árbol de búsqueda.
- Propiedades:
  - *Completa sólo si* $L \geq d$ (*d* profundidad de la solución más somera)
    - Si *d* es desconocido, ¿cómo elegir *L*?
      - En general no se conoce a priori.
      - Utilizar conocimiento específico del dominio.

      Ej. Hay problemas en los que se conoce el *diámetro* del espacio de estados. Dicho diámetro es el número máximo de pasos necesarios para alcanzar cualquier estado desde cualquier otro estado. Por lo tanto, se puede utilizar *L* = *diámetro*, lo que garantiza encontrar una solución, si esta existe.
  - No es óptima: No puede garantizarse que, en caso de que encuentre una solución, esta sea la de menor coste.
  - Complejidad temporal: $O(b^L)$ [exponencial en *L*]
  - Complejidad espacial: $O(b*L)$ [Lineal en *L*] (buenas noticias)

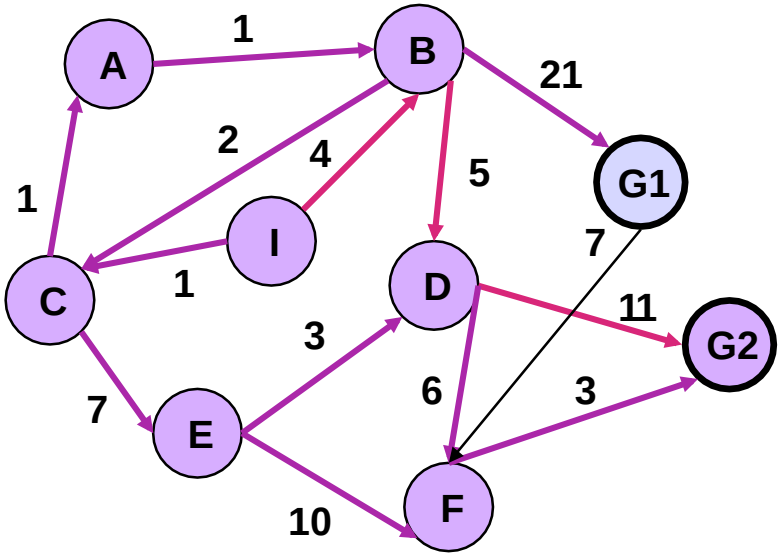
Búsqueda primero en profundidad con límite de profundidad

- Límite $L = 5$
  - Eliminación de estados repetidos
  - Sólo expandidos

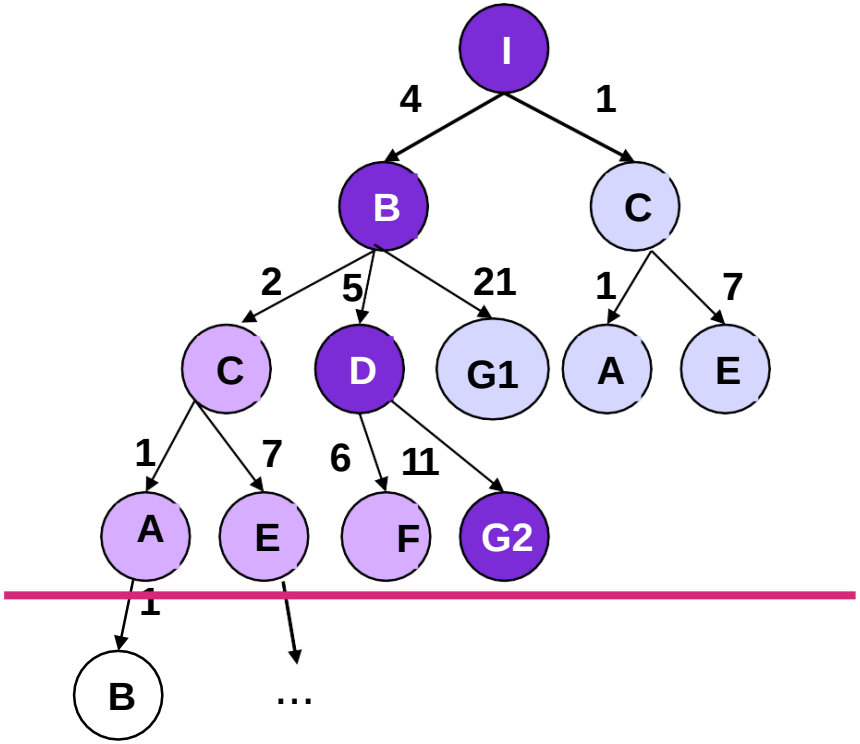


Ej. Búsqueda primero en profundidad con límite de profundidad *(L = 3)*

Espacio de estados



Espacio de búsqueda



Nodos expandidos (por orden de expansión: I B C A E D F G2

Camino a la solución: I B D G2 Coste: 4+5+11 = 20

Búsqueda con vuelta atrás (backtracking)

Variante de la búsqueda con profundidad limitada, con **uso eficiente de la memoria**

- Sólo se genera un sucesor en cada expansión. En cada nodo parcialmente expandido se recuerda cuál es el siguiente sucesor a generar.

  $\Rightarrow O(m)$ estados

- Generar sucesor modificando el estado actual (en vez de copiar + modificar). Se deben poder deshacer modificaciones cuando se vaya hacia atrás para generar los siguientes sucesores.

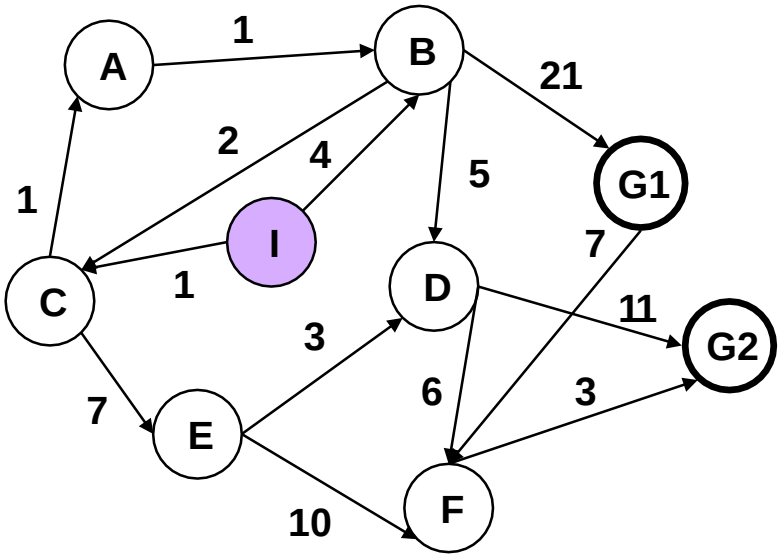
  $\Rightarrow$ un solo estado + $O(m)$ acciones

Búsqueda en profundidad con límite de profundidad incremental

- Aplicación iterativa del algoritmo de búsqueda de profundidad limitada, con límite de profundidad creciente (L = 0,1, 2,...)
  - Evita tener que determinar el límite de profundidad
- Combina las ventajas de los algoritmos primero en profundidad y primero en anchura.
- Suele ser el método no informado preferido cuando el espacio de estados es grande y no se conoce la profundidad de la solución.
- Propiedades:
  - *Completa:* garantiza encontrar una solución (la menos profunda de entre las posibles)
  - *Óptima:* solo si el coste del camino es una función no decreciente de la profundidad del nodo (el camino de menor longitud puede no ser óptimo)
  - Tiempo: $O(b^d)$ *(Puede ser mejor que primero en anchura)*
    - Nodos generados: b^d * *(1 vez)* + ... + b^2 * *(d-2)* + b^1 * *(d-1)* + b^0 * *(d veces)*
    - Los nodos que se generan de manera repetida son los que se encuentran en niveles de profundidad más bajos, pero evitamos generar nodos a profundidad *(d+1)*, lo que, en el peor de los casos son b^{d+1} operaciones.
  - Espacio: $O(b*d)$ (como el algoritmo de búsqueda primero en profundidad)

Ejemplo: búsqueda con profundización iterativa

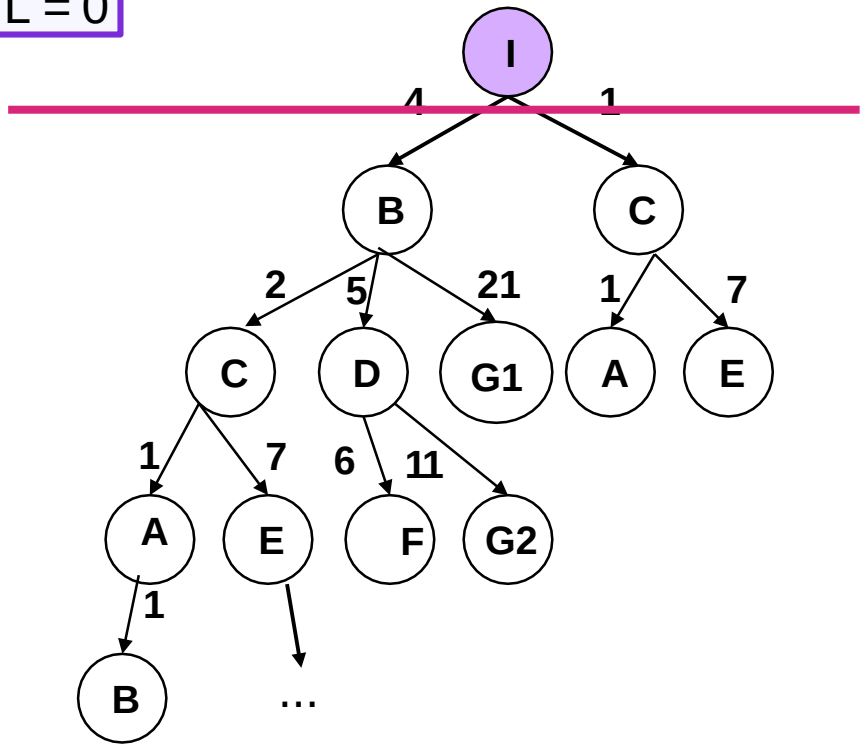
Espacio de estados



Nodos expandidos: I

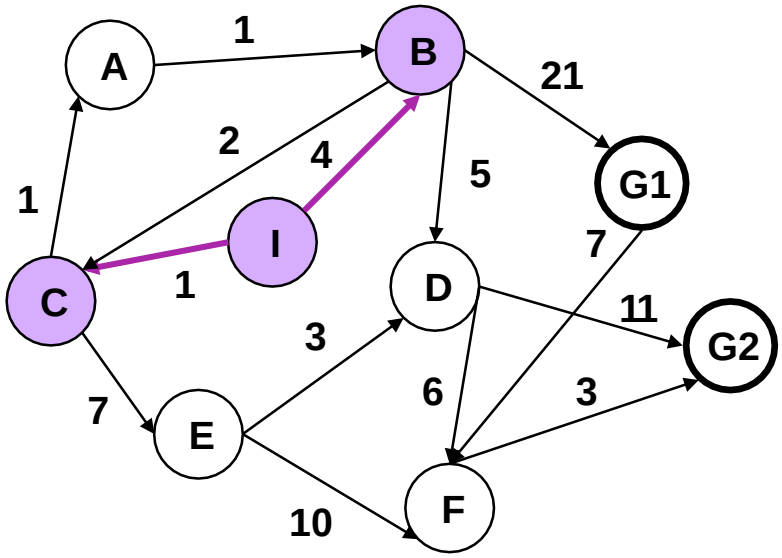
Espacio de búsqueda

L = 0



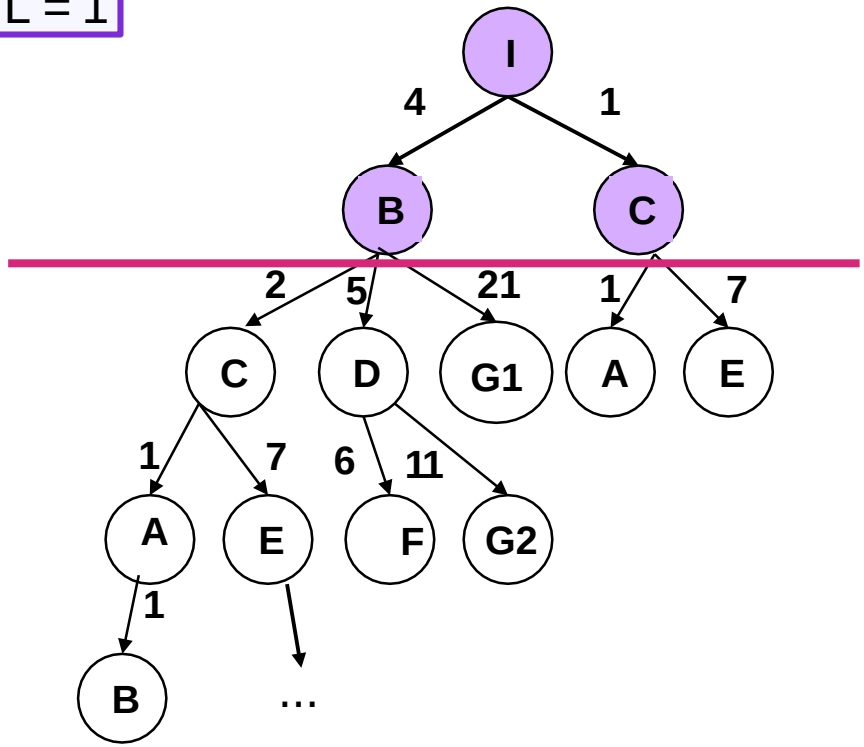
Ejemplo: búsqueda con profundización iterativa

Espacio de estados



Espacio de búsqueda

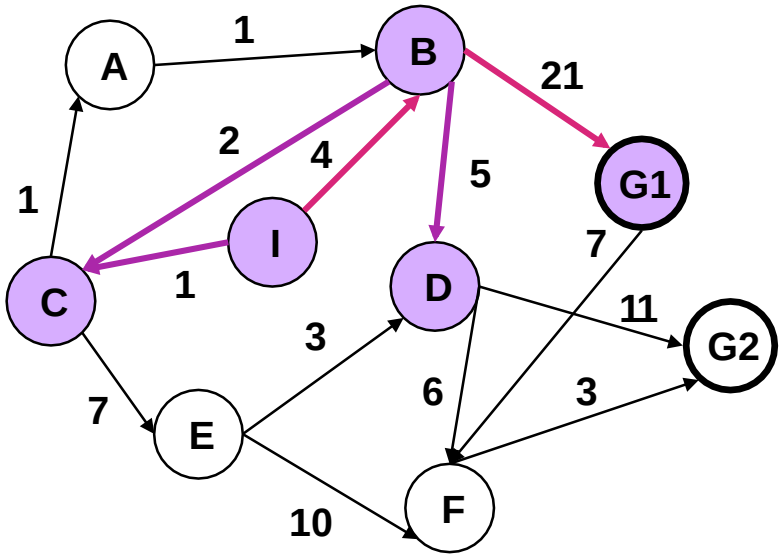
L = 1



Nodos expandidos: I B C

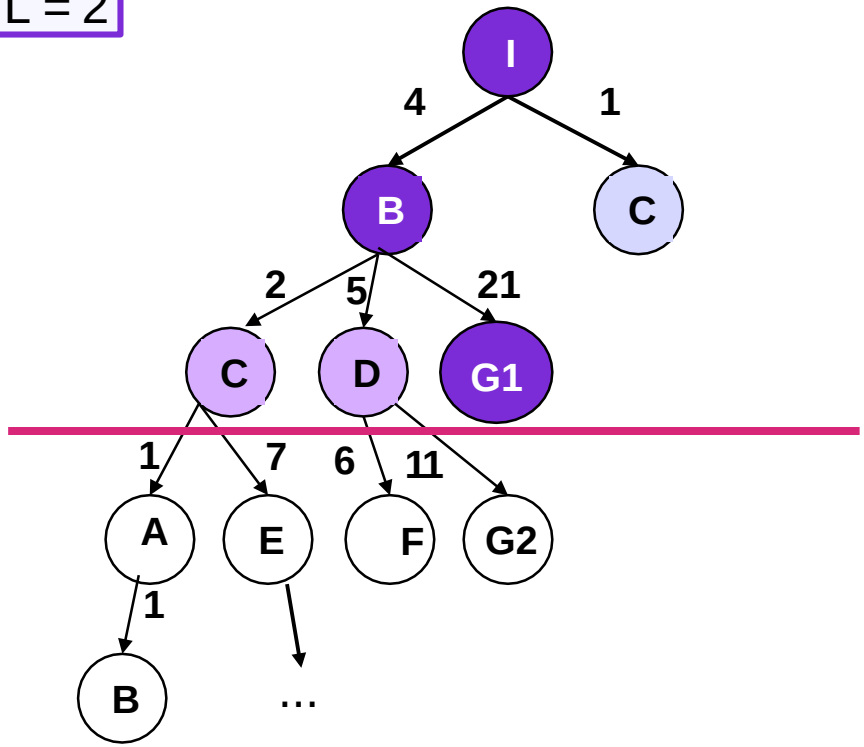
Ejemplo: búsqueda con profundización iterativa

Espacio de estados



Espacio de búsqueda

L = 2



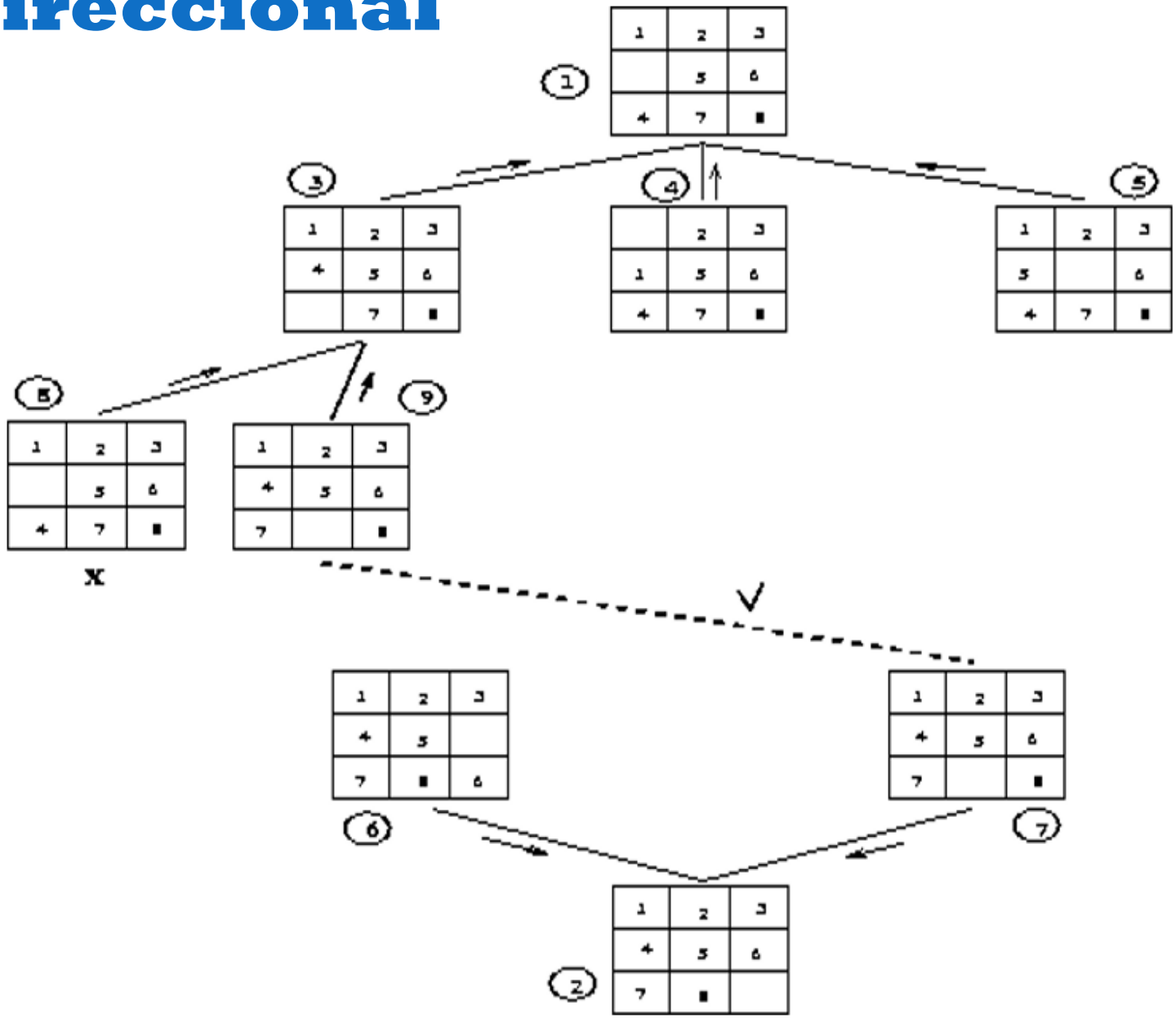
Nodos expandidos: I B C I B C D G1

Camino a la solución: I B G1 Coste: 4+21 = 25

Búsqueda bidireccional

- Ejecuta dos búsquedas simultáneas: una hacia delante desde el estado inicial y la otra hacia atrás desde el estado objetivo, parando cuando las dos búsquedas se encuentren en un estado
  - Motivación: $b^{d/2} + b^{d/2}$ es mucho menor que b^d
  - Es necesario
    - Conocer el estado objetivo. Si hay varios estados, o estos no se pueden listar (por ejemplo, en el problema de N reinas), esto puede ser problemático
    - Poder obtener los predecesores de un estado de una manera efinciante (operadores invertibles.)
- Propiedades:
  - *Óptima y completa* si las búsquedas son en anchura el coste del camino es una función no decreciente de la profundidad del nodo.
  - Tiempo: $O(2*b^{d/2}) = O(b^{d/2})$
    - Si la comprobación de la coincidencia puede hacerse en tiempo constante (por ejemplo, mediante una tabla hash)
  - Espacio: $O(b^{d/2})$ *(su mayor debilidad)*
    - Al menos, los nodos de una de las dos partes se deben mantener en memoria para la comparación *(suele usarse una tabla hash para guardarlos)*

Búsqueda bidireccional



-- Resumen de métodos no informados --

BÚSQUEDA CIEGA	Completa	Óptima	Eficiencia Tiempo (caso peor)	Eficiencia Espacio (caso peor)
Primero en Anchura	Sí	Si coste $\propto$ profundidad	$O(b^d)$	$O(b^d)$
Coste uniforme	Si no hay caminos de coste finito y longitud infinita	Si coste operadores > 0	$O(b^{C^*/\varepsilon})$	$O(b^{C^*/\varepsilon})$
Primero en profundidad	No	No	$O(b^m)$	$O(b*m)$
Profundidad limitada	Si $L \geq d$	No	$O(b^L)$	$O(b*L)$
Profundización iterativa	Sí	Si coste $\propto$ profundidad	$O(b^d)$	$O(b*d)$
Bidireccional con búsqueda en anchura	Sí (anchura)	Si coste $\propto$ profundidad	$O(b^{d/2})$	$O(b^{d/2})$

Factor de ramificación (***b***): número máximo de sucesores de cualquier nodo.
Profundidad del nodo objetivo más superficial (***d***)
Profundidad máxima del árbol de búsqueda (***m***)
Límite de profundidad (***L***)
Coste mínimo de una acción (ε)
Coste camino solución óptima (C*)

Tipos de problema: con información parcial

Hasta ahora hemos analizado problemas de búsqueda en los que

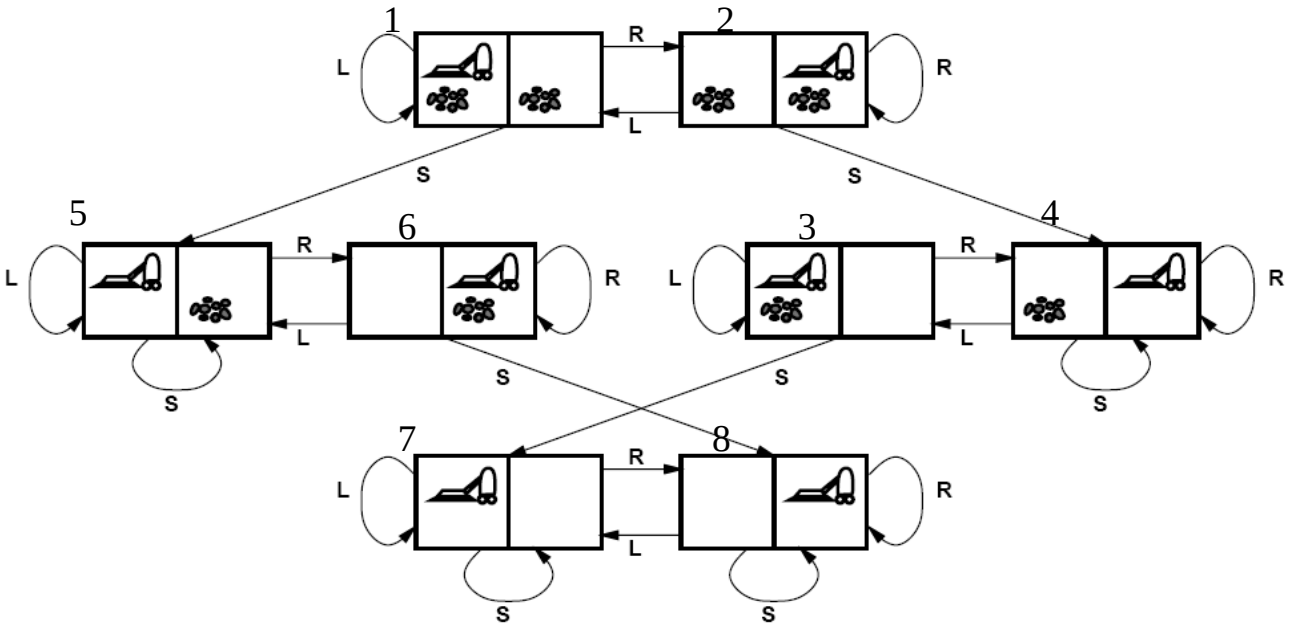
- Espacio de estados
  - **completamente observable** (las percepciones determinan unívocamente el estado del entorno)
  - **estático** (únicamente cambia cuando ejecutamos una acción).
- **Acciones deterministas:** Conocemos el resultado de cualquier secuencia de acciones.

¿Que ocurre cuando no disponemos de información completa para caracterizar un estado de manera unívoca?

- **Problemas de búsqueda conformante:** No observable (problemas **sin sensores**). El agente no dispone de información completa sobre el estado en el que se encuentra.
- **Problemas de contingente**
  - No determinista, y posiblemente parcialmente observable. Las percepciones del agente proporcionan **nueva** información después de cada acción.
  - Si la **incertidumbre** la causa las acciones de otro agente con objetivos opuestos a los del agente que realiza la búsqueda => búsqueda con **adversarios**
- **Problemas de exploración**: Espacio de estados desconocido. Es necesario interactuar con el entorno (sensores y efectores) para determinar en qué estado se encuentra.
- **Problemas de búsqueda contingente con exploración**
  - Los algoritmos **estándar de búsqueda no son, en general, adecuados** para resolver este tipo de problemas.
  - El agente puede recabar **información** a través de sus sensores **después de actuar**.
  - Proposición: **alternar búsqueda y ejecución (acciones + sensores).**

Mundo de la aspiradora

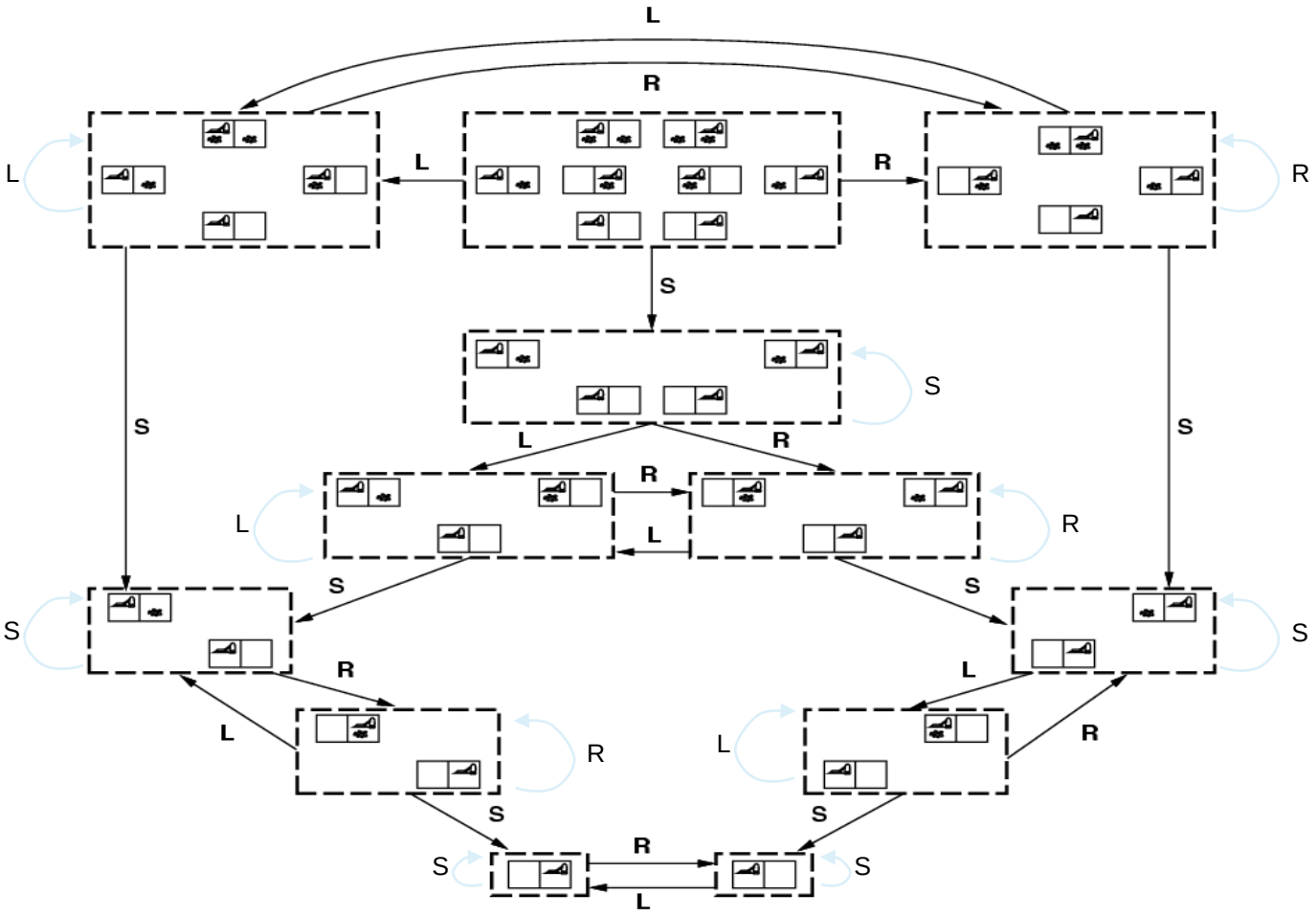
- Estados físicos:
  - Agente: Aspiradora en uno de las dos estancias adyacentes.
  - Cada estancia puede estar sucia / limpia
- Acciones:
  - L / R: mover a la izquierda / derecha
  - S: Aspirar suciedad
  - NoOp: No hacer nada
- Objetivo: Mundo limpio (estado 7 u 8)



Problema conformante (sin sensores)

Si el agente **no tiene sensores**, entonces

- Los estados de búsqueda representan **estados de creencia: subconjuntos** de los posibles estados físicos del mundo.
- A través de una **secuencia de acciones** uno puede ser capaz de **forzar** al mundo a un estado objetivo.
- Sin sensores no sabemos si la acción tiene efecto o no.



Problema de búsqueda contingente

- La solución no puede ser formulada en general como una secuencia fija de acciones.
- **Plan de contingencia**: La solución puede ser dada como un árbol, donde se elige una rama u otra dependiendo de las percepciones

Ejemplo:

- Estado sencillo, se empieza en estado #5.
  ¿Solución?
  [*Right*, *Suck*]

- Conformado, se empieza en {1, 2, 3, 4, 5, 6, 7, 8}
  Por ejemplo, la acción *Right* conduce al estado {2, 4, 6, 8}. **¿Solución?**
  [*Right*, *Suck*, *Left*, *Suck*]

- Contingencia, se empieza en estado #5.
  El aspirador tiene un fallo y utilizarlo puede hacer que el suelo se ensucie.
  Sensores: localización, suciedad. **¿Solución?**
  [*Right*, **if** *suelo-sucio* **then** *Suck*]