Tema 2.2: Búsqueda en el espacio de estados

- Espacio de búsqueda
- Esquema general de búsqueda
- Tipos de búsqueda
- Búsqueda ciega
  - Primero en anchura
  - Coste uniforme
  - Primero en profundidad
  - Profundidad limitada
  - Profundización iterativa
  - Bidireccional
  - Técnica del museo británico

Material desarrollado por distintos autores del departamento de ISIA

Resolución de problemas

- *Generate and test*
  - Relación con el proceso cognitivo humano.. Los humanos lo hacemos constantemente porque no tenemos conocimiento completo y correcto del mundo.
  - No tenemos mucha capacidad de computo.
  - Generadores y tester “listos” no es fuerza bruta.
  - Dominios de planificación más complejos: sistemas multiobjetivo, interacción y dependencia de las acciones (las dependencias entre acciones) y conflictos entre objetivos.
- Relación también con algoritmos genéticos
  - Mutación y cruce son *generate*.
  - Fitness es el *testing*.

Espacio de búsqueda (I)

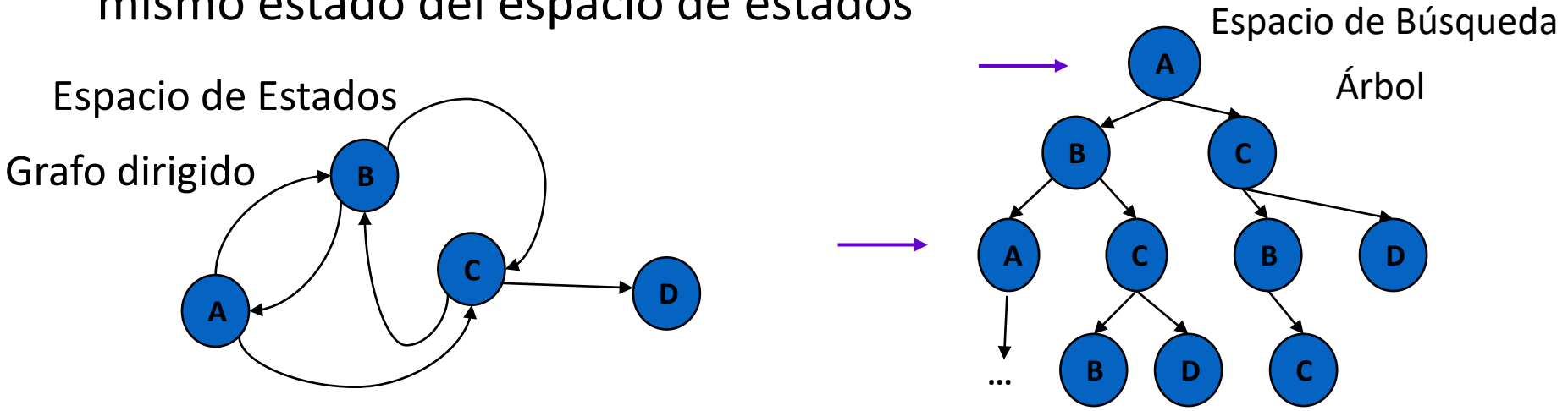
- El espacio de búsqueda (o árbol de búsqueda) solo contiene los nodos generados en la búsqueda de la solución

Resolución de problemas con búsqueda en espacio de estados:

- No supone que el grafo esté previamente generado
- Ni asume tampoco que se tengan que generar todos los estados → imprescindible para el tipo de problemas de IA
  - El espacio de estados del 8-puzzle tiene 9!/2 = 181.440 estados...
- Sólo se generan los nodos necesarios según la estrategia de búsqueda

Espacio (o árbol) de búsqueda

- Aunque el espacio de estados sea finito (puede no serlo),…el espacio de búsqueda puede ser infinito por los posibles ciclos del grafo.
- Dos nodos distintos del árbol de búsqueda pueden referirse al mismo estado del espacio de estados



- Tipo de datos **NODO** del árbol de búsqueda:

Estado
puntero al nodoPadre (y opcional qué operador)
profundidad
coste acumulado

Espacio de búsqueda. Control de repeticiones

- Evitar la repetición de estados tiene un coste
- Puede hacerse a distintos niveles
  1. Evitar aplicación sucesiva de operadores inversos *(si hay)* → bajo coste
  2. Evitar ciclos en el camino actual (guardando estados del camino actual)
  3. Evitar la repetición de un estado en cualquier camino → puede requerir mucho espacio y tiempo
  - Debe de haber un compromiso entre lo que se intenta evitar y el coste de evitarlo → cuanto más ciclos, más justificado

Esquemas generales de búsqueda

Frontera = Nodos abiertos
Explored set = Nodos cerrados

```
function TREE-SEARCH(problem) returns a solution, or failure
    initialize the frontier using the initial state of problem
    loop do
        if the frontier is empty then return failure
        choose a leaf node and remove it from the frontier
        if the node contains a goal state then return the corresponding solution
        expand the chosen node, adding the resulting nodes to the frontier
```

```
function GRAPH-SEARCH(problem) returns a solution, or failure
    initialize the frontier using the initial state of problem
    initialize the explored set to be empty
    loop do
        if the frontier is empty then return failure
        choose a leaf node and remove it from the frontier
        if the node contains a goal state then return the corresponding solution
        add the node to the explored set
        expand the chosen node, adding the resulting nodes to the frontier
            only if not in the frontier or explored set
```

Figure 3.7 An informal description of the general tree-search and graph-search algorithms. The parts of GRAPH-SEARCH marked in bold italic are the additions needed to handle repeated states.

Esquema general de búsqueda (TreeSearch)

(libro Rusell & Norvig pág. 77)

Tree search (sin control de repeticiones)
(frontier = abiertos)

function TREE-SEARCH(problem) returns a solution, or failure

Initialize the frontier using the initial state of the problem

loop do

if *the frontier is empty* then return *failure*

choose a leaf node and *remove it from the frontier*

if the node contains a goal state then **return** the corresponding solution

expand the chosen node, adding the resulting nodes to the frontier

La elección dependerá de la estrategia

Esquema general de búsqueda (GraphSearch)

- **Graph search (con control de repeticiones)**
  (explored set = cerrados)

function GRAPH-SEARCH(problem) returns a solution, or failure

initialize the frontier using the **initial state of problem**

initialize the **explored set** to be empty

loop do

if the **frontier is empty** then return failure

choose a leaf node and remove it from the frontier

if the node contains a goal state then return the corresponding solution

ANTES DE EXPANDIR EL NODO

add the node to the explored set

expand the chosen node, adding the **resulting nodes** to the frontier only if not in the frontier or explored set

ANTES DE METER EL NODO EN LA LISTA FRONTERA

Elementos del esquema general de búsqueda

- **Nodos Abiertos** (o **frontera**): Nodos pendientes de expandir
- **Nodos Cerrados:** Nodos ya expandidos o visitados (explored-set)

Un nodo se expande una única vez

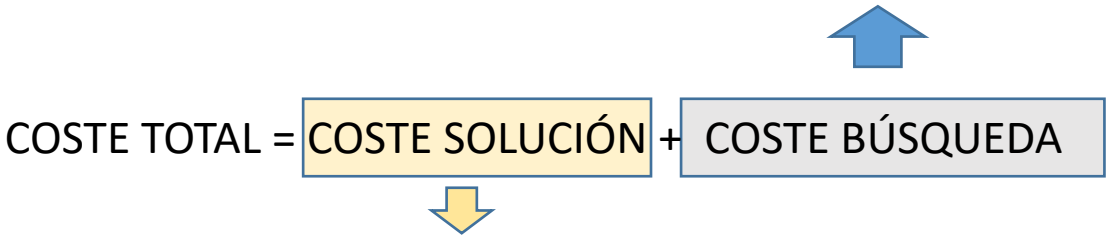
- Tipos de **estados**:
  - **Generados**: Son aquéllos que aparecen o han aparecido en nodos de *abiertos (abiertos U cerrados)*
  - **Generados pero no expandidos**: Son aquéllos que aparecen en la lista de nodos de *abiertos*
  - **Expandidos**: Un estado se expande cuando
    1. Un nodo que lo representa se quita de *abiertos*,
    2. Se generan todos sus descendientes y éstos se añaden a *abiertos*

Un estado puede ser expandido varias veces (en diferentes nodos)

Evaluación de las estrategias de búsqueda

- **Completitud**: ¿garantiza encontrar solución **si la hay**?
- **Optimalidad**: ¿encuentra la **solución de coste mínimo**?

- **Complejidad** en **tiempo**: ¿cuánto tarda en encontrar una solución en el peor caso ?→ nº de nodos expandidos
- **Complejidad** en **espacio**: ¿cuánta memoria se necesita en el **peor** caso? ¿cuál es el máximo nº de nodos simultáneamente en memoria?

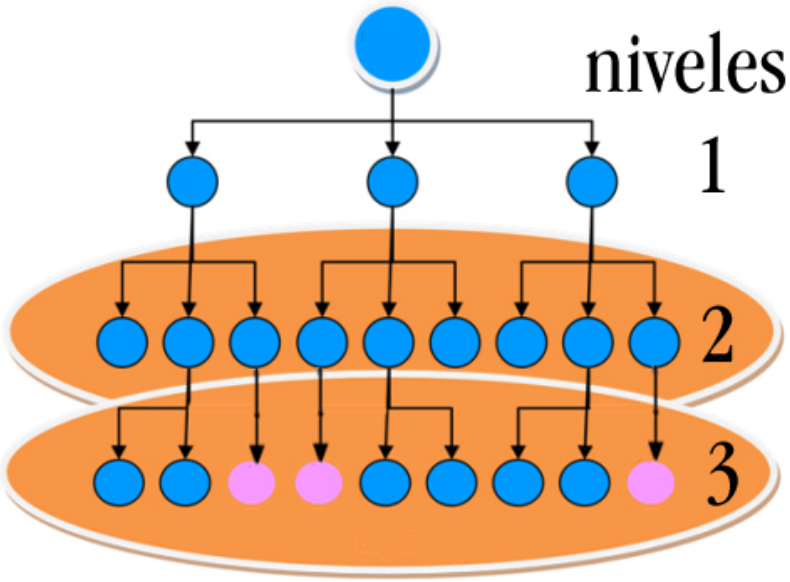


Métodos de búsqueda

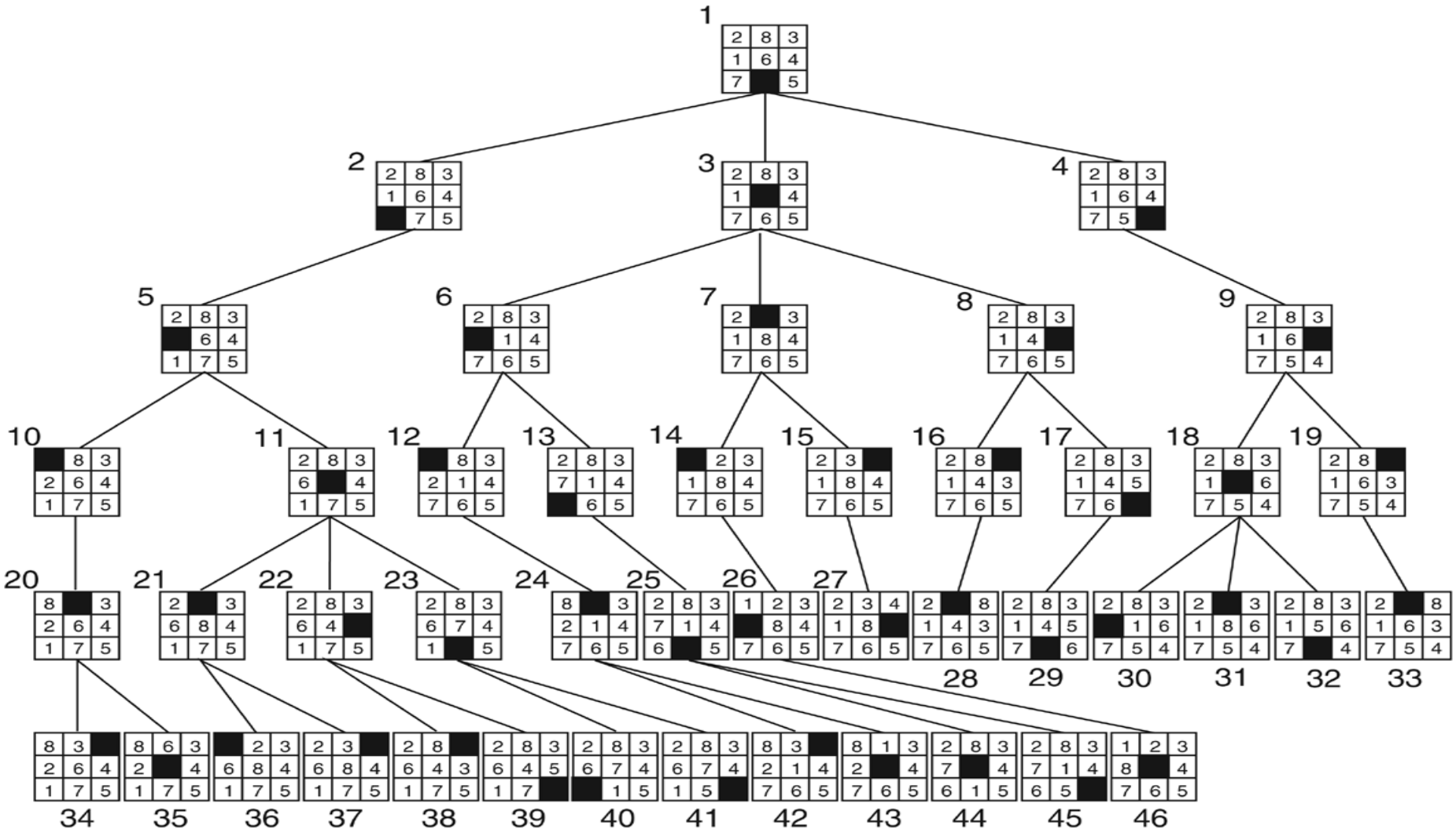
- **Métodos no informados o ciegos**:
  - Exploración *exhaustiva* del espacio de búsqueda hasta encontrar una solución
  - No incorporan conocimiento que guíe la búsqueda. Se decide a priori qué camino sigue
  - La búsqueda no incorpora información del dominio
- **Métodos informados o heurísticos**:
  - Exploración de los caminos más *prometedores*
  - Se incorpora conocimiento del dominio → Funciones Heurísticas
  - Las Funciones Heurísticas son "pistas" para acotar el proceso de búsqueda y hacerlo más eficiente

Búsqueda primero en anchura

- Se explora el espacio de búsqueda haciendo un recorrido por niveles.
- Un nodo se visita solo cuando todos sus predecesores y sus hermanos anteriores en orden de generación ya se han visitado
- **La estructura abiertos del algoritmo general se implementa con una COLA (FIFO)** → Para visitar los nodos en el orden establecido por el algoritmo de búsqueda



Árbol de la búsqueda primero en anchura para el 8-puzzle



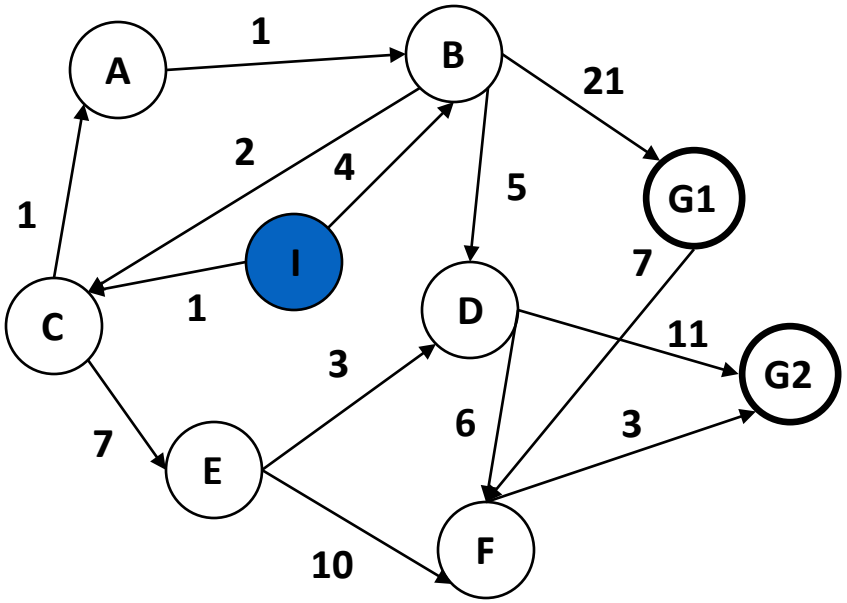
Búsqueda primero en anchura. Propiedades

- Es un algoritmo **completo** → Su sistematicidad nos garantiza llegar al nivel donde se encuentra la solución (**si la hay**)
- Complejidad en tiempo: $O(r^p)$
  - Suponiendo un factor de ramificación *máximo* ***r*** (nº de hijos de un nodo) y un camino hasta la solución de profundidad ***p*** → el nº de nodos expandidos en el caso peor es $r^0 + r^1 + r^2 + \ldots + r^p$
- Complejidad en espacio: $O(r^p)$
  - Todos los nodos abiertos (pendientes de explorar) han de mantenerse en memoria → Sólo viable para casos pequeños
  - Sistema potente actual y unos valores de r= 10 y p=8
    → 11 Gbytes y 31 horas => sólo aplicable a problemas de juguete
- Si los operadores de búsqueda tienen coste uniforme la solución obtenida es óptima respecto al número de pasos desde la raíz

Búsqueda primero en anchura. Ejemplo

Espacio de búsqueda

Espacio de estados

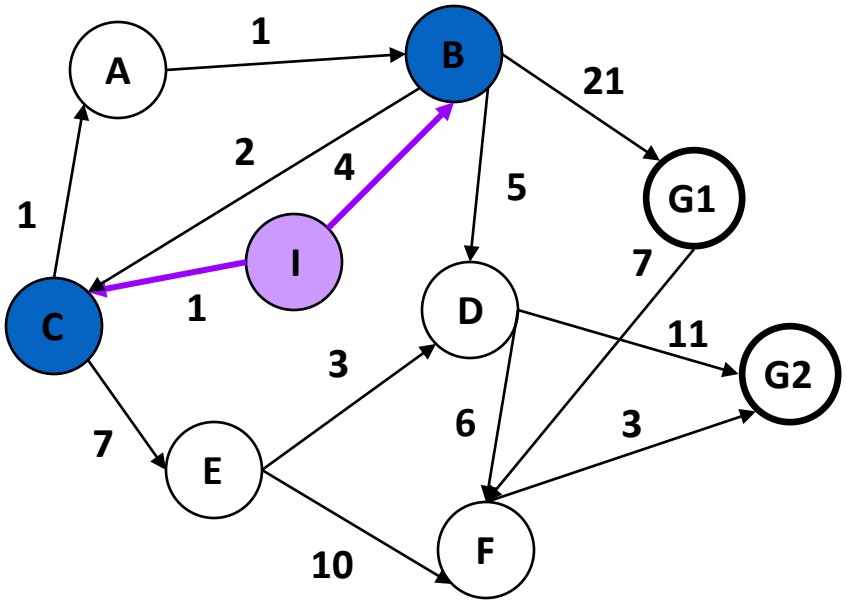


¿Cuál de los dos G se alcanzará?

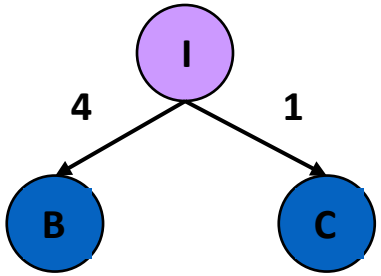
Cola de nodos abiertos: I

Búsqueda primero en anchura. Ejemplo

Espacio de estados



Espacio de búsqueda



Frente de la cola

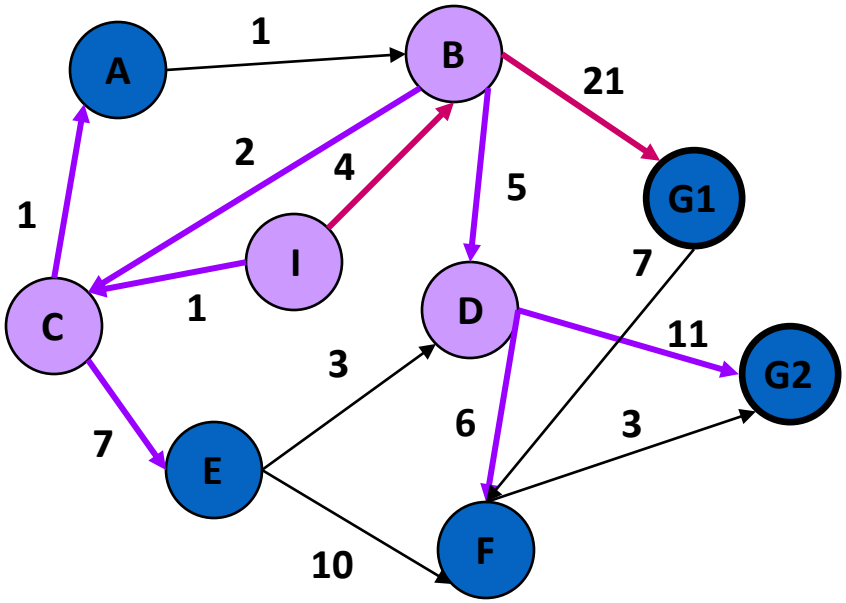
↓

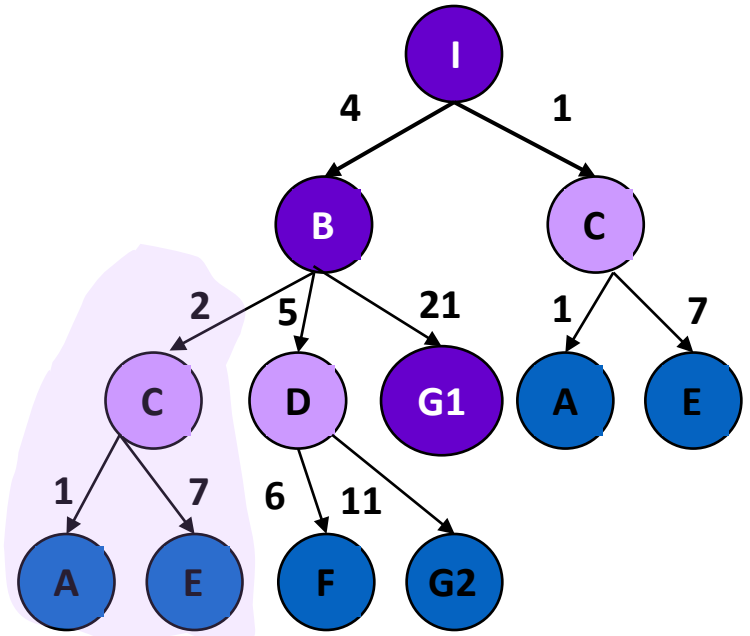
Cola de nodos abiertos: C B

Búsqueda primero en anchura. Ejemplo

Espacio de búsqueda

Espacio de estados





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

¿No era óptimo? ¿y el camino de coste 17 a G2 (I-C-A-B-D-F-G2) ?

Búsqueda de coste uniforme

- En cada paso, de los nodos en *abiertos*, se expande el nodo que tiene el **menor coste del camino** hasta llegar a él
  - Si hay varios iguales se elige aleatoriamente o de izquierda a derecha
- *abiertos* se implementa con una cola de prioridad
- Si el coste del camino a un nodo coincide con su profundidad (o es directamente proporcional a la profundidad) entonces esta búsqueda equivale a primero en anchura

Búsqueda de coste uniforme. Propiedades

- **Completa** si no existen caminos infinitos de coste finito
- **Óptima** si *coste(sucesor(n)) >= coste(n)* → Se satisface cuando todos los operadores **tienen *coste >= 0***
- Complejidad en espacio y tiempo equivalente a primero en anchura: $O(r^p)$

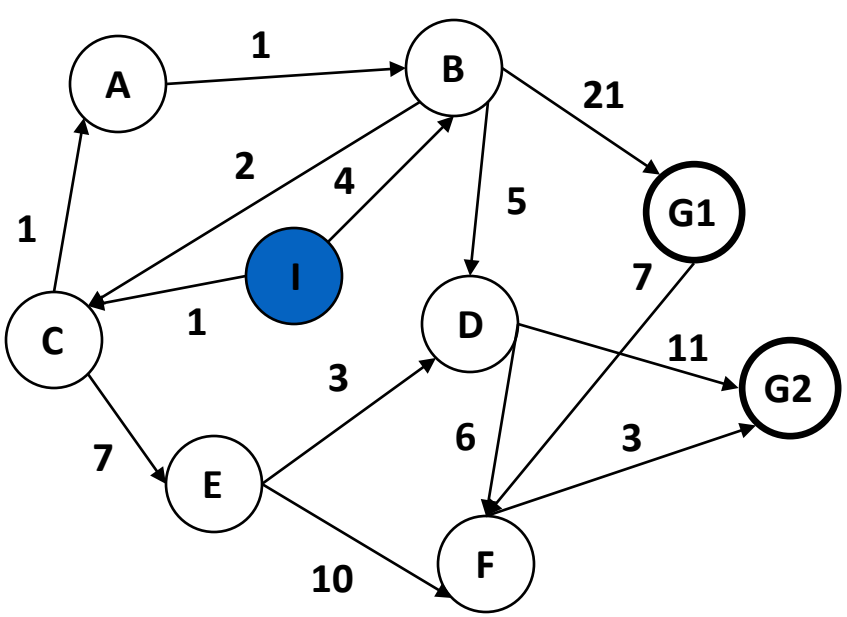
Algoritmo de Dijkstra es una variante de coste uniforme sin estado objetivo

Otras variantes como el algoritmo de Bellman-Ford permiten costes < 0

Búsqueda de coste uniforme. Ejemplo

Espacio de búsqueda

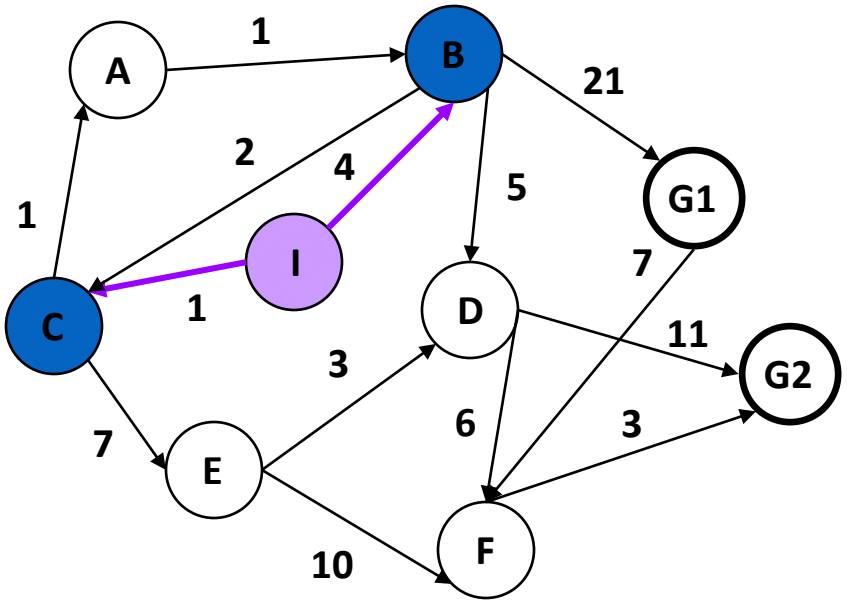
Espacio de estados



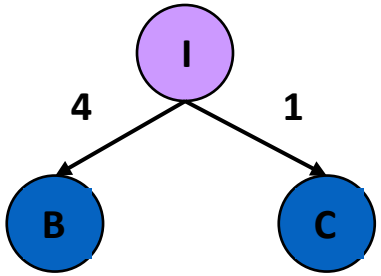
Cola de prioridad de nodos abiertos: I(0)

Búsqueda de coste uniforme. Ejemplo

Espacio de estados



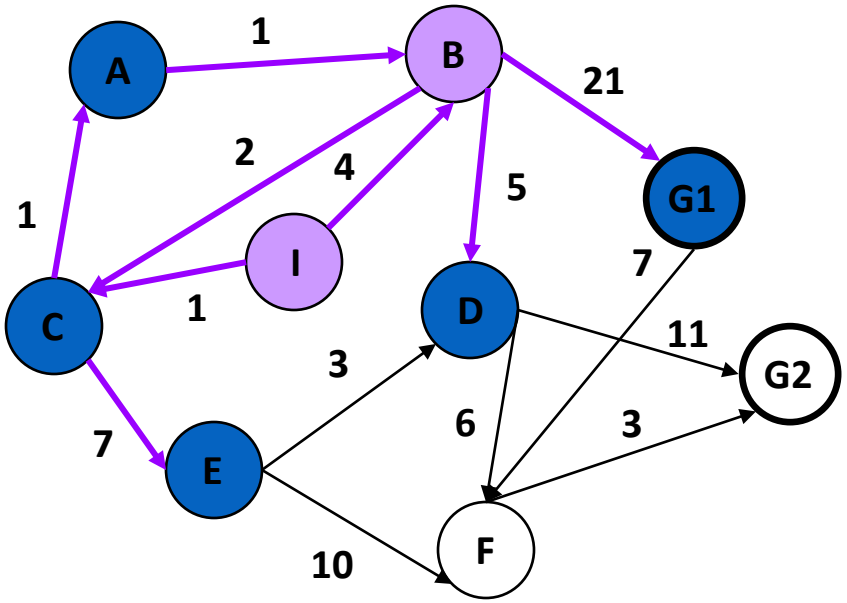
Espacio de búsqueda



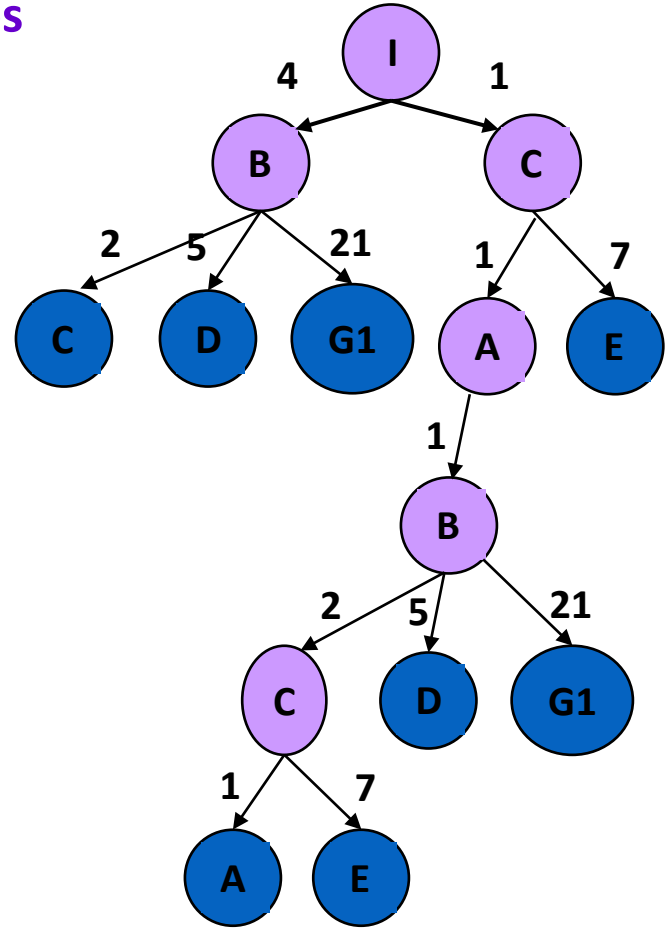
Cola de prioridad de nodos abiertos: B(4) C(1)

Búsqueda de coste uniforme. Ejemplo

Espacio de estados



Espacio de búsqueda

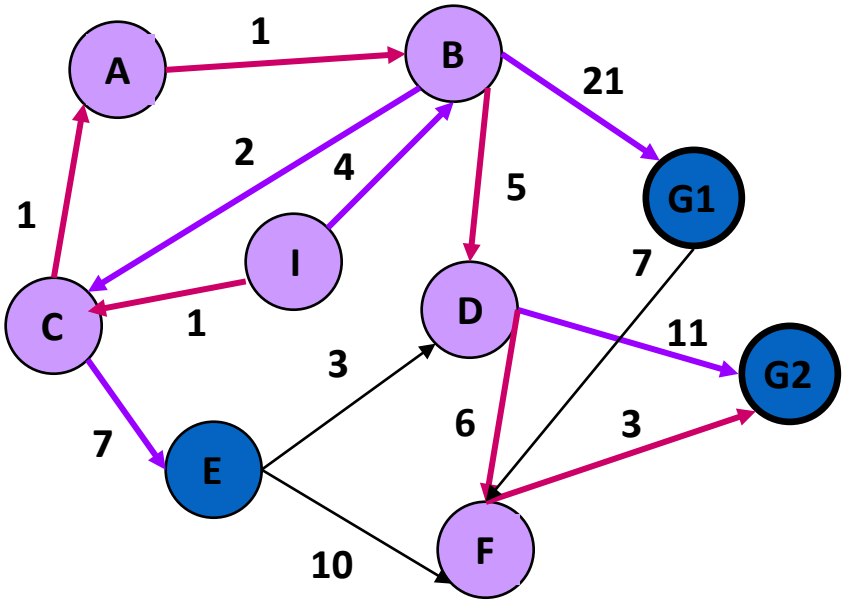


Y así seguiría unos cuantos pasos más...

Cola de prioridad de nodos abiertos: G1(25) G1(24) E(12) D(9) D(8) E(8) A(6) C(6)

Búsqueda de coste uniforme. Ejemplo

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. Repeticiones

- El problema de las repeticiones es mucho mayor
- *Habría que evitar repeticiones sólo si coste > anterior*
- Un control indiscriminado de repeticiones 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 de repeticiones sólo sobre la rama actual si funcionaría*

Búsqueda primero en profundidad

- Los nodos se expanden por orden inverso de generación (Estructura de pila - LIFO)
- El comportamiento en el esquema general:
  - Se extrae como nodo actual el último que entró en *abiertos*
  - Expandir nodo actual y quitar de abiertos
    - Si no es objetivo y no tiene sucesores
  - El siguiente nodo que se expande es el más profundo de la lista de abiertos → esto es la vuelta atrás
- La estructura *abiertos* se implementa con una **pila** (o usa recursión)

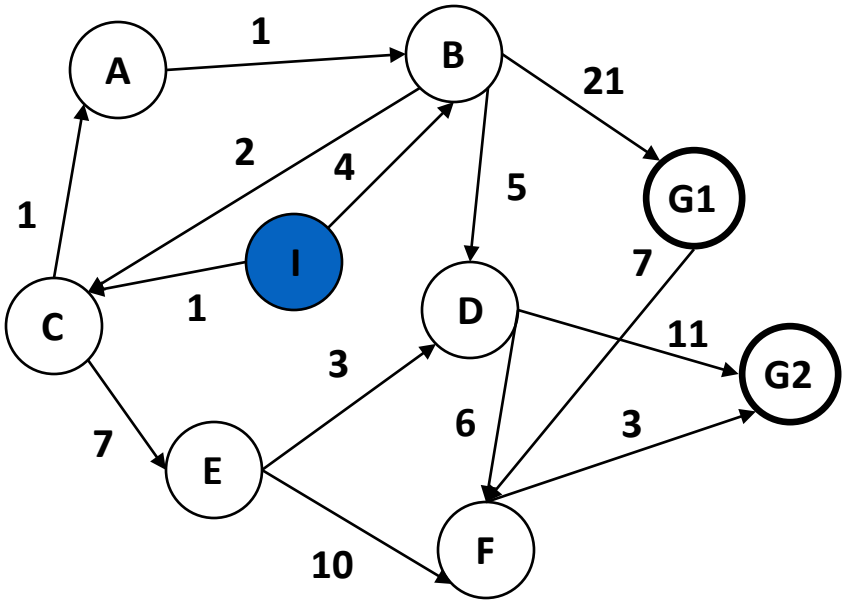
Búsqueda primero en profundidad. Propiedades

- **No completa**: puede meterse en caminos infinitos
- **No óptima**: puede haber soluciones mejores por otros caminos
  - ➢ No recomendable si la máxima profundidad es grande
- Complejidad en espacio: **O(r*m)**, siendo r el factor de ramificación y m la profundidad máxima
  - Basta guardar el camino actual y los nodos no expandidos (frontera)
- Tiempo: **$O(r^m)$**
  - Si hay muchas soluciones, puede ser más rápida que primero en anchura; depende del orden de aplicación de operadores

Búsqueda primero en profundidad. Ejemplo

Espacio de búsqueda

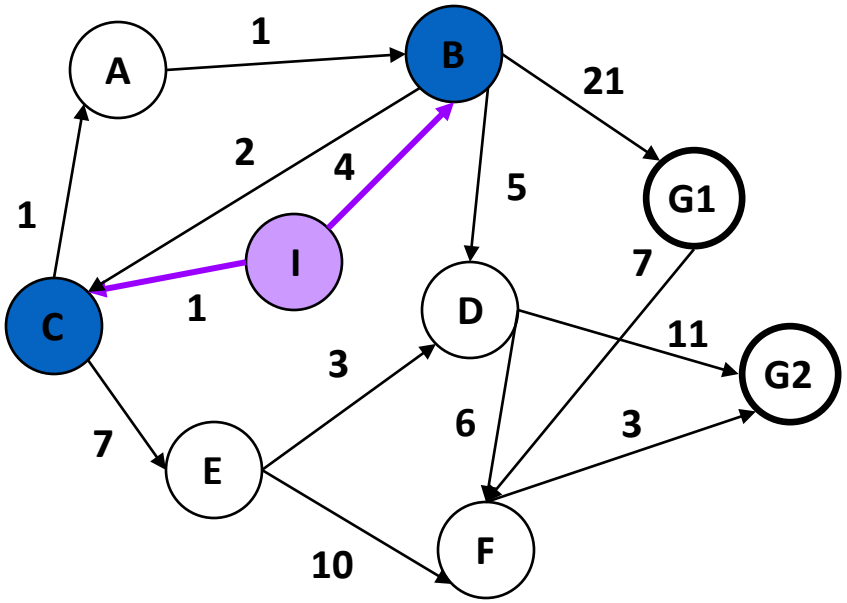
Espacio de estados



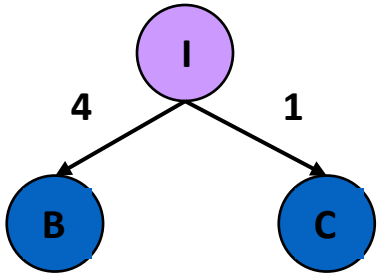
Pila de nodos abiertos: I

Búsqueda primero en profundidad. Ejemplo

Espacio de estados



Espacio de búsqueda

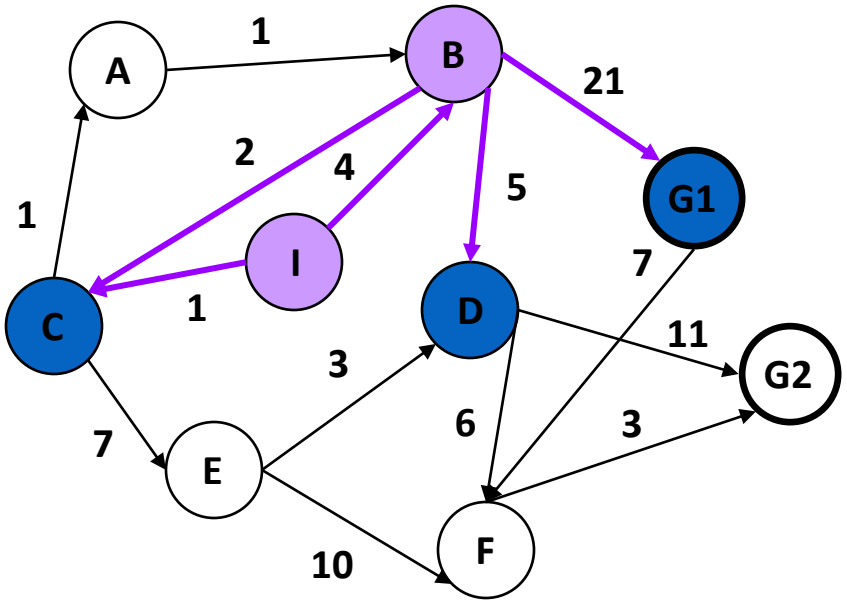


Cabeza de pila

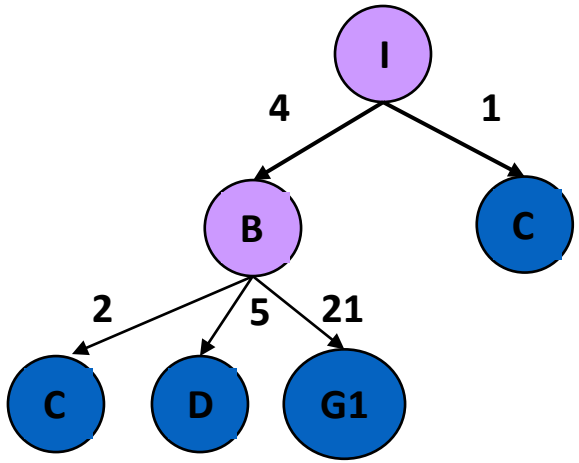
Pila de nodos abiertos: B C

Búsqueda primero en profundidad. Ejemplo

Espacio de estados



Espacio de búsqueda



Cabeza de pila

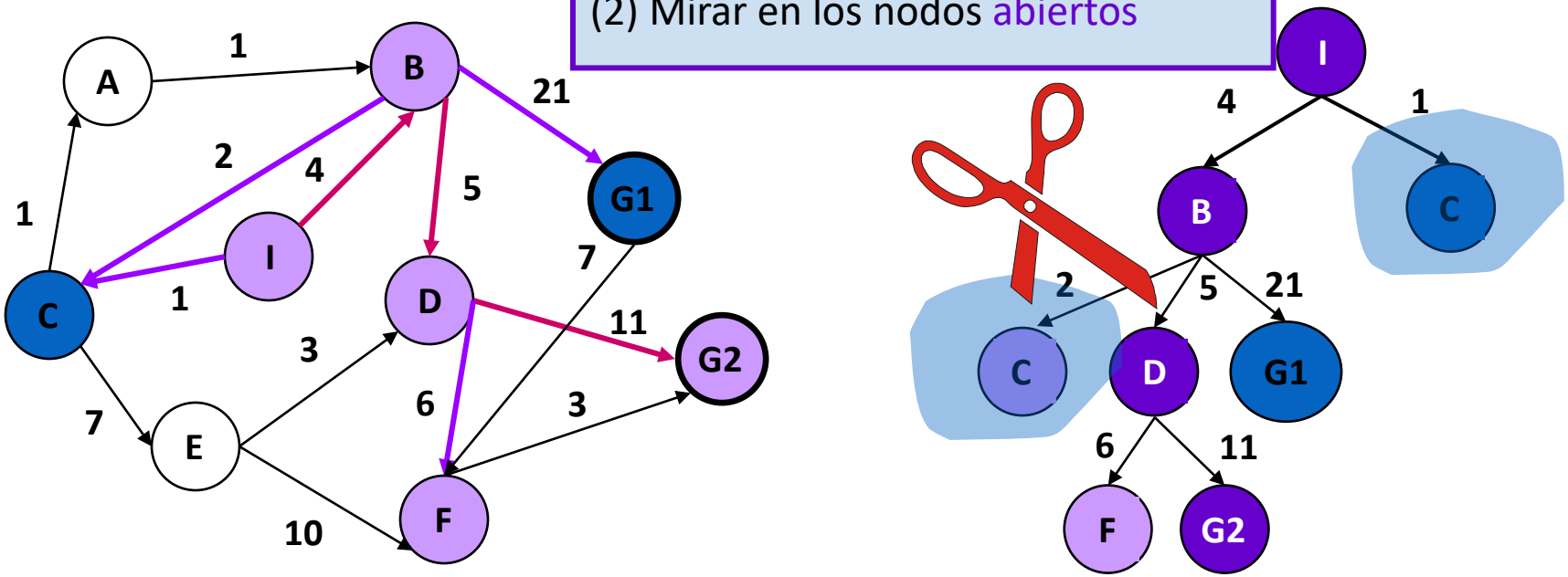
Pila de nodos abiertos: C D G1 C

Búsqueda primero en profundidad. Ejemplo

Espacio de búsqueda

Espacio de estados

(2) Mirar en los nodos abiertos



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

Nodos abiertos (por orden): I B C D G1 F G2

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

Búsqueda primero en profundidad. Ventajas

- Necesita menos memoria → Sólo se almacenan, en esencia, los nodos del camino que se sigue, mientras que en anchura se almacena la mayor parte del árbol que se va generando

- Con "suerte" puede encontrar una solución sin tener que examinar gran parte del espacio de estados (depende en gran medida de que el orden de expansión sea "adecuado“)
  - Es muy dependiente del orden de aplicación de los operadores
  - Encuentra la solución “más a la izquierda”

Búsqueda de profundidad limitada

- Es la búsqueda en profundidad con límite L de profundidad para evitar descender indefinidamente por el mismo camino

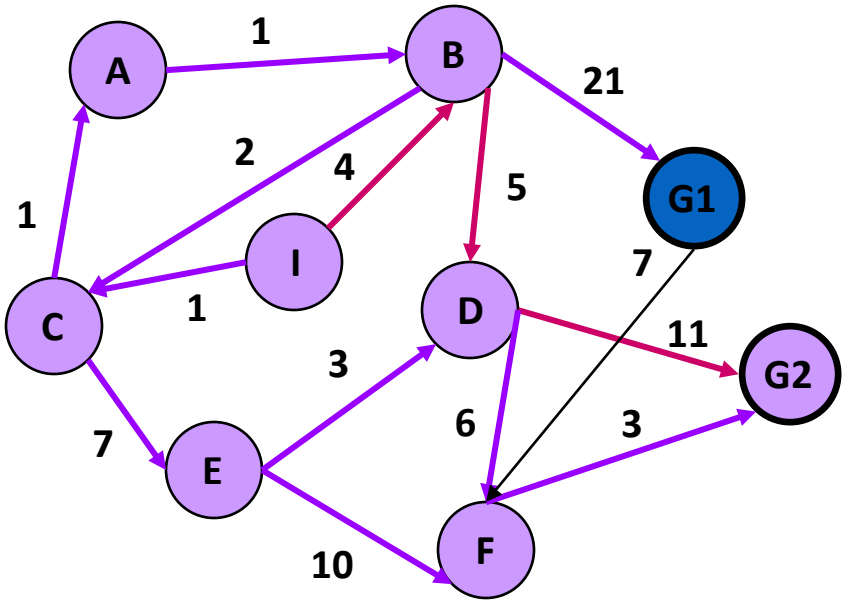
  - El límite permite desechar caminos en los que se supone que no encontraremos un nodo objetivo lo suficientemente cercano al nodo inicial

Búsqueda de profundidad limitada. Propiedades

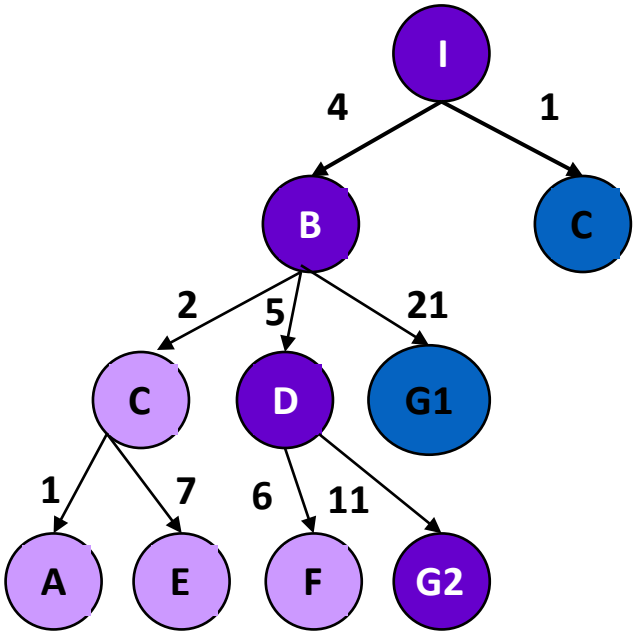
- ***Completa sólo si L >= p*** (*p* profundidad mínima de "la solución óptima")
  - Si *p* es desconocido, la elección de *L* es una incógnita
    - ➢ Hay problemas en los que este dato (diámetro del espacio de estados) es conocido de antemano, pero en general no se conoce a priori
- **No óptima** → No puede garantizarse que la primera solución encontrada sea la mejor, sólo la de más a la izquierda.
- Ventaja respecto a búsqueda en profundidad → evitamos el coste del control de repeticiones
- Tiempo: $O(r^L)$
- Espacio: $O(r*L)$

Búsqueda de profundidad limitada. Ejemplo (L=3)

Espacio de estados



Espacio de búsqueda



Nodos expandidos: I B C A E D F G2

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

Búsqueda de profundidad iterativa

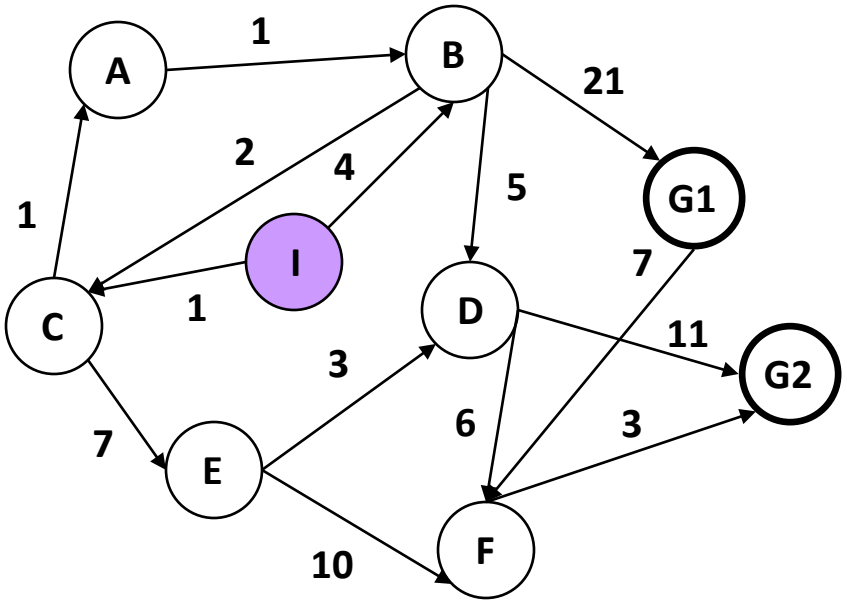
- Aplicación iterativa del algoritmo de búsqueda de profundidad limitada, con límite de profundidad variando de forma creciente (0,1,...) → Evita el problema de la elección del límite
- Combina las ventajas de los algoritmos primero en profundidad y primero en anchura
- Suele ser el método **no informado** preferido cuando el espacio de estados es grande y la profundidad de la solución se desconoce

Búsqueda de profundidad iterativa. Propiedades

- **Completo** y **óptimo** (como búsqueda en anchura)
- Tiempo: $O(r^p)$ (algo peor que primero en anchura)
  - Nodos expandidos: $r^p * 1$ vez + ... + $r^2 * (p-1) + r^1 * p + r^0 *$ (p+1 veces)
  - La repetición de cálculos afecta principalmente a niveles superiores por lo que no es excesivamente importante
- Espacio: O(r*p) (como primero en profundidad)
- El tratar repetidos acaba con todas las ventajas espaciales del algoritmo
  - Sólo sería posible control en el camino actual

Búsqueda de profundidad iterativa. Ejemplo

Espacio de estados



Espacio de búsqueda

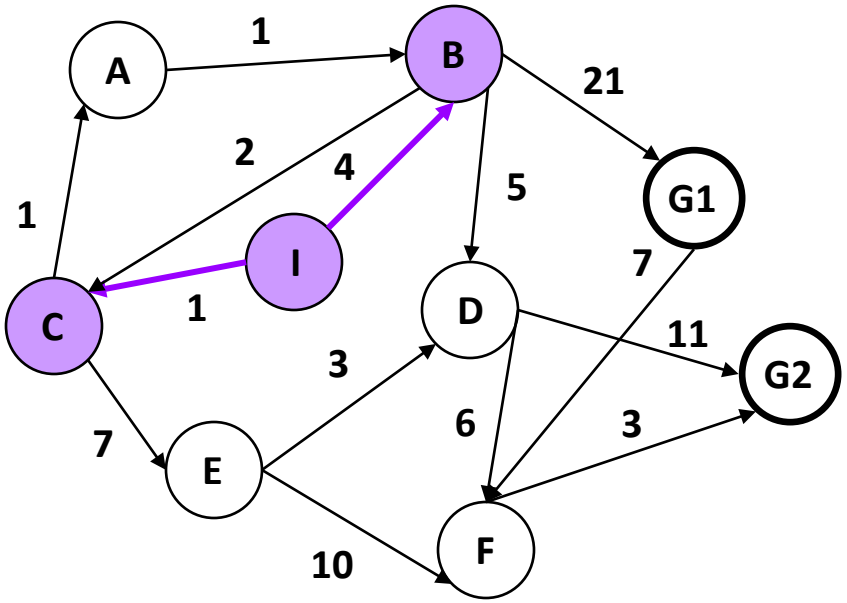
L = 0

I

Nodos expandidos: I

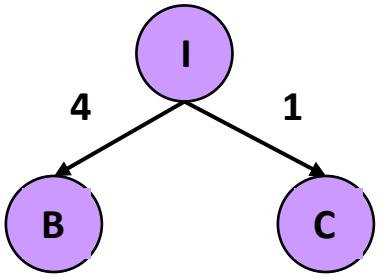
Búsqueda de profundidad iterativa. Ejemplo

Espacio de estados



Espacio de búsqueda

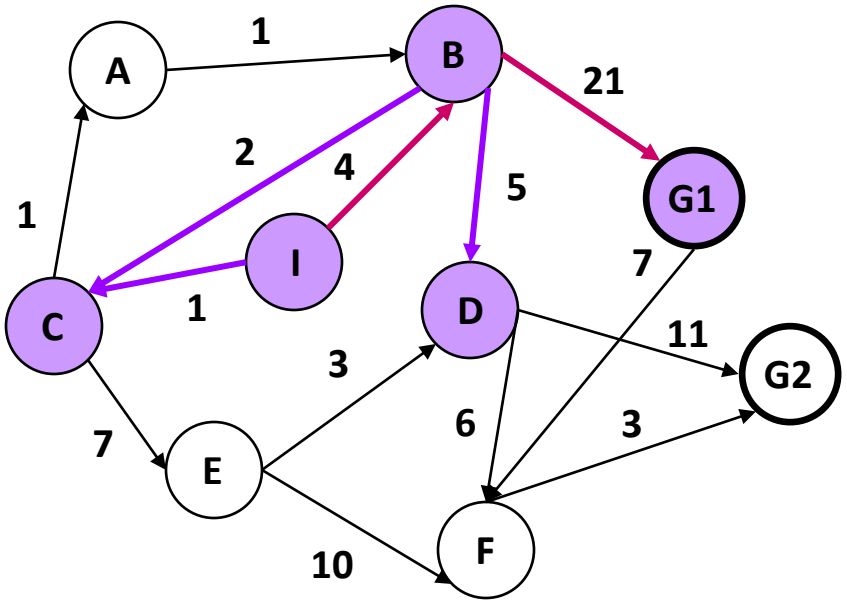
L = 1



Nodos expandidos: I I B C

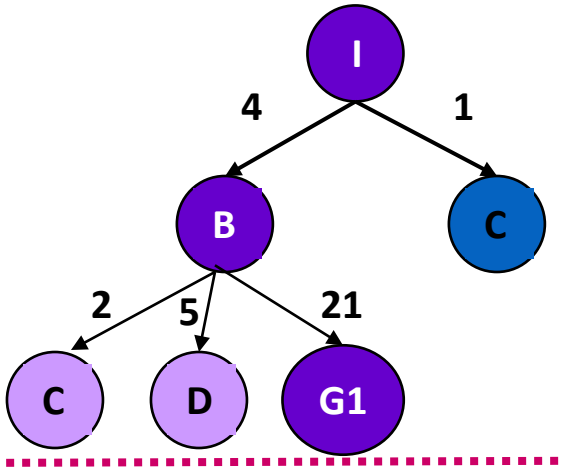
Búsqueda de profundidad iterativa. Ejemplo

Espacio de estados



Espacio de búsqueda

L = 2



Nodos expandidos: I I B C I B C D G1

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

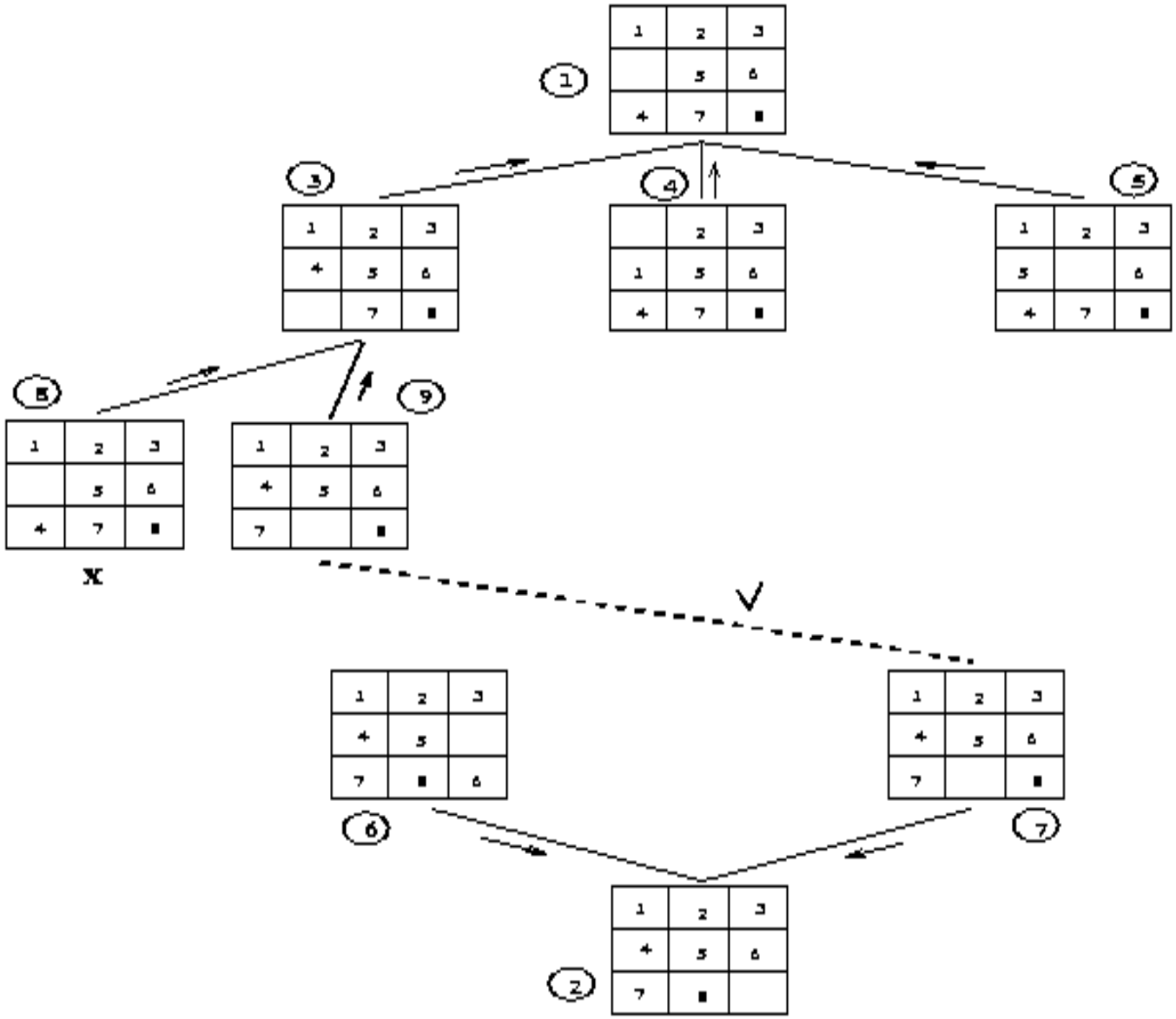
Búsqueda bidireccional

- 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 mismo estado
- Motivación: $r^{p/2} + r^{p/2}$ es mucho menor que r^p
- Es necesario
  - Conocer el estado objetivo (si hay varios, puede ser problemático)
  - Poder obtener los predecesores de un estado (operadores reversibles)
- No está fijado qué algoritmos de búsqueda habría que utilizar en ambos lados
  - Los más lógico es que en uno de ellos se trabaje con búsqueda en anchura
  - En el otro mejor un algoritmo con profundización iterativa para que no haya que mantener en memoria todos los nodos

Búsqueda bidireccional. Propiedades

- Óptima y completa si las búsquedas son en anchura y los costes son iguales en todos los operadores
  - ➢Otras combinaciones no lo garantizan: ¡pueden no encontrarse!
- Tiempo: $O(2*r^{p/2}) = O(r^{p/2})$
  - ➢Si la comprobación de la coincidencia puede hacerse en tiempo constante
- Espacio: $O(r^{p/2})$
  - Al menos, los nodos de una de las dos partes deben mantenerse en memoria para la comparación
    - ➢Suele usarse una tabla hash para guardarlos
  - Es la mayor debilidad del algoritmo

Búsqueda bidireccional. Ejemplo



Bibliografía

- **Russell, S., Norvig, P.** Artificial Intelligence. A Modern Approach. Prentice Hall, 2013, 3ª edición.
- Curso IA con Python (Universidad Harvard) https://video.cs50.io/D5aJNFWsWew?screen=2wNUOxYQhdI
- https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-034-artificial-intelligence-fall-2010/lecture-videos/lecture-4-search-depth-first-hill-climbing-beam/ Lección 4