3.-Resolución de problemas y espacio de búsqueda

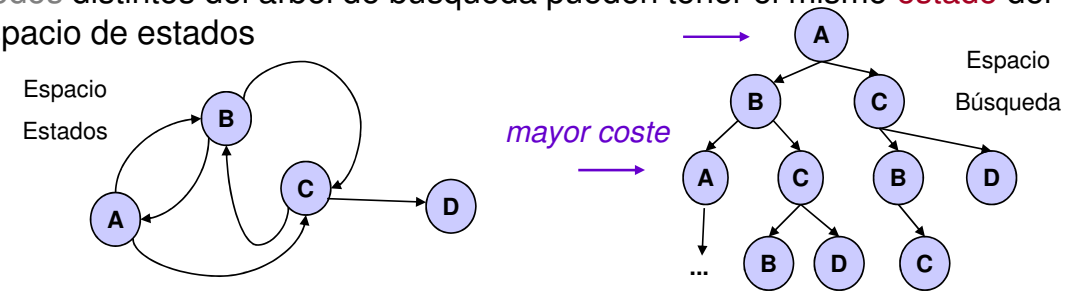
- **2. Búsqueda**
  - Introducción
    - Espacio de búsqueda
    - Esquema general de búsqueda
    - Evaluación de estrategias de búsqueda
  - Métodos no informados o ciegos
  - Métodos informados o heurísticos

2. Búsquedas en el Espacio de Estados

- Es un mecanismo general de resolución de problemas
  - Tenemos los estados y los operadores, pero…
  - Qué operador aplicar en cada momento?
  - Cómo sé que me acerco a la solución?
- Dos tipos de Búsquedas
  - 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 p.e.: primero en profundidad
    - La búsqueda no incorpora información del dominio
  - Métodos informados o heurísticos
    - Exploración de los caminos más *prometedores*
    - Se incorpora conocimiento del dominio: Funciones Heurísticas
      - "pistas" para acotar el proceso de búsqueda y hacerlo más eficiente

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 por los posibles ciclos del grafo
  - Nodos distintos del árbol de búsqueda pueden tener el mismo estado del espacio de estados



  - Un nodo 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)

Espacio de búsqueda y Ciclos

- Evitar la repetición de estados (ciclos del grafo) 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 ningún camino
     => mayor coste espacio-tiempo para hacer las comparaciones
     - Marcar los estados que han sido generados para comprobar la no repetición
- Un compromiso entre lo que se intenta evitar y el coste de evitarlo
  - cuanto más ciclos, más justificado
- Búsqueda en espacios de estados: (a diferencia de teoría algorítmica de grafos)
  - 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 grafo del 8-puzzle tiene 9! = 362.880 vértices...
  - Sólo se generan estados necesarios de acuerdo a la estrategia

Esquema general de búsqueda

```
función BÚSQUEDA_GENERAL (problema, estrategia)
   devuelve solución o fallo      % o cicla
     inicializar abiertos con                    % … generados sin expandir
            nodo(estado_inicial del problema)    % generación nodo
     repetir
          si abiertos = Ø
            entonces devolver fallo
          extraer nodo de abiertos según estrategia
                                % el orden de extracción lo determina la estrategia
          si nodo contiene estado_objetivo
            entonces devolver solución
            si no                               % aquí se controlaría si repetidos
                  expandir nodo                 % generar los nodos hijos
                  añadir todos sus hijos a abiertos
                                % orden de aplicación de operadores (generación)
```


Terminología

- Estructura Abiertos
  - Se guardan nodos con estados generados pendientes de expandir
- Estructura Cerrados
  - Se guardan los ya expandidos o visitados (solo en algunos algoritmos)
- Tipos de estados/nodos
  - Generados
    - Son aquéllos que aparecen o han aparecido en nodos de *abiertos*
  - Generados pero no expandidos
    - Son aquéllos que aparecen en 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)
      - Un nodo se expande una única vez

Evaluación de estrategias de búsqueda

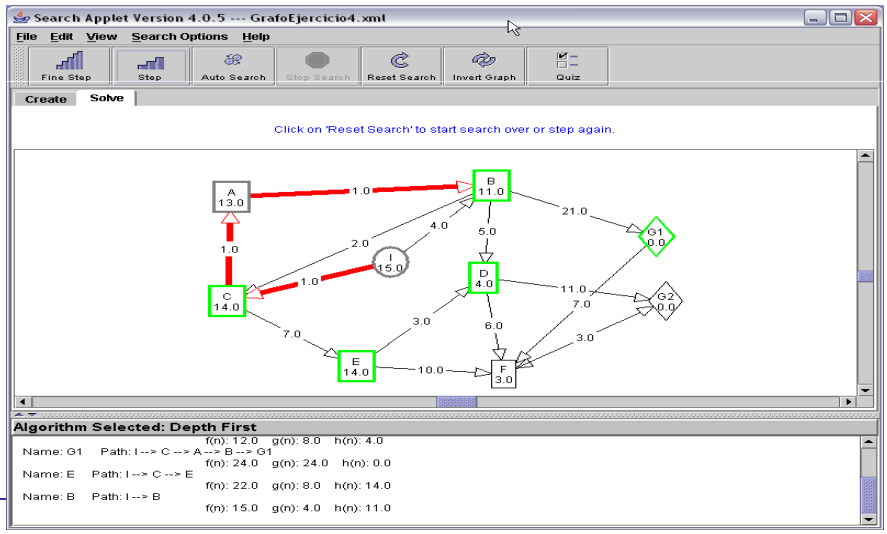
- Criterios para la evaluación de estrategias de búsqueda
  - Completitud: ¿garantiza encontrar solución si la hay?
  - Optimalidad: ¿encuentra la solución de coste mínimo?
  - Complejidad en tiempo: ¿cuánto tarda en encontrar una solución? Es el peor caso: el nodo objetivo el último del árbol (nº de nodos expandidos)
  - Complejidad en espacio: ¿cuánta memoria se necesita en el peor caso? (máximo nº de nodos simultáneamente en memoria)
- Coste total: 2 componentes
  - Coste de la solución: suma del coste operadores usados en el camino
    - a veces, en problemas de optimización, coste de la solución en sí misma
  - Coste de la búsqueda en sí: complejidad del algoritmo
- Hay que llegar a un compromiso entre ambos costes
  - Obtener la mejor solución posible con los recursos disponibles
    - No quiero una solución buena pero que lleva dos días calcularla (muy cara)

Resolución de problemas y espacio de búsqueda

- **Métodos no informados o ciegos**
  - Primero en anchura
  - Coste uniforme
  - Primero en profundidad
  - Profundidad limitada
  - Profundización iterativa
  - Bidireccional
- **Ver simulaciones en www.aispace.org/search**

Evolucion Algoritmos http://aispace.org/search/

- Este ejemplo Teoria.xml : se abre con "File" + "load from file"
- Para ver los costes y heurística: "view" + "show node heuristics"
- Se puede ver la evolución del algoritmo: solapa "solve"
  paso a paso "step" o "fine step"
- Criterio de desempate: menor coste (no el orden alfabético)
  - Se puede cambiar en: “Search Options” + “Neighbor Ordering Strategies”

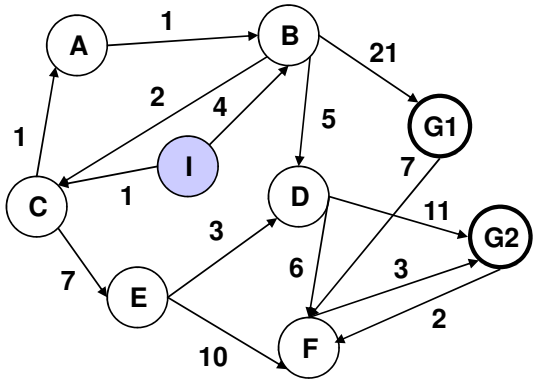


---Búsqueda primero en anchura (Moore, 1959)---

- Los nodos se expanden: por orden no decreciente de *profundidad*
  - Nodos de profundidad *p* se expanden antes que los nodos de profundidad *p+1*
    - *(por lo que no hay vuelta atrás)*
  - La estructura *abiertos* se implementa con una cola
- Propiedades (criterios de evaluación)
  - Completa y *óptima si el coste del camino es función no decreciente de la profundidad del nodo (el camino de menor longitud puede no ser óptimo)*
  - Complejidad en tiempo y en espacio: $O(r^p)$
    - Suponiendo un factor de ramificación *máximo* ***r*** (nº de hijos de un nodo) y
    - un camino hasta la solución de profundidad mínima ***p***,
    - el nº de nodos expandidos en el caso peor es $r^0 + r^1 + r^2 + ... + r^p + (r^{p+1} - r)$
  - Problema mayor: la memoria (sólo viable para casos pequeños)
    - Todos los nodos generados han de mantenerse en memoria
- Ciclos: no se pierden soluciones, aunque suponen ineficiencia
  - Si no hay solución, podrían no terminar

Ejemplo: búsqueda primero en anchura

Espacio de estados

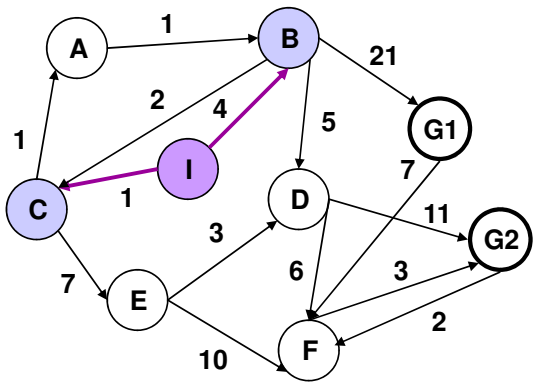


Espacio de búsqueda

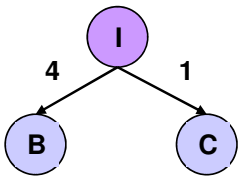
Cola de nodos abiertos: I

Ejemplo: búsqueda primero en anchura

Espacio de estados



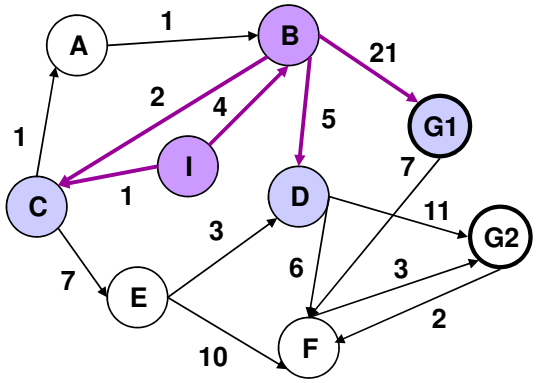
Espacio de búsqueda



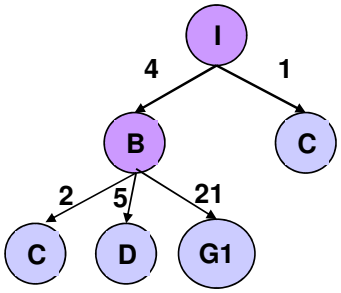
Cola de nodos abiertos: C B

Ejemplo: búsqueda primero en anchura

Espacio de estados



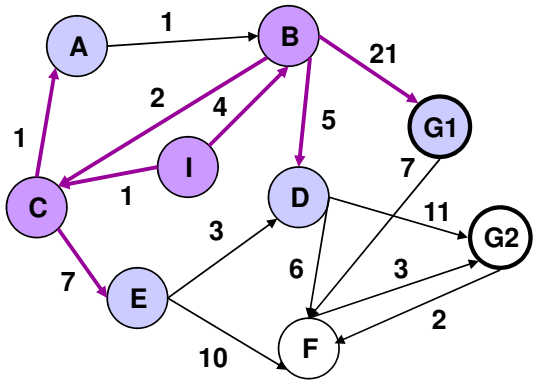
Espacio de búsqueda



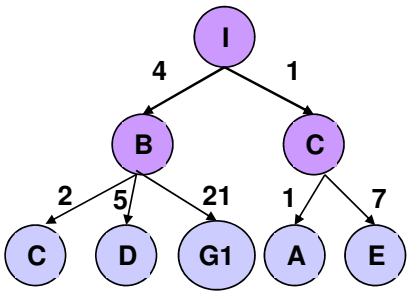
Cola de nodos abiertos: G1 D C C

Ejemplo: búsqueda primero en anchura

Espacio de estados



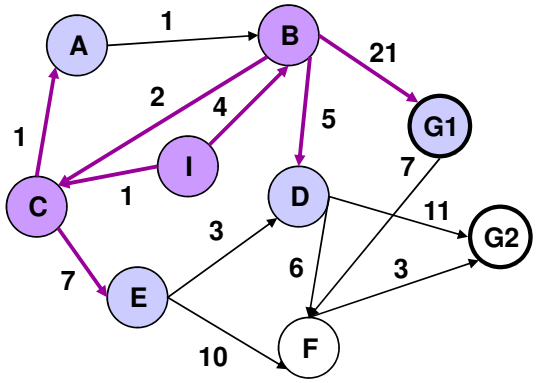
Espacio de búsqueda



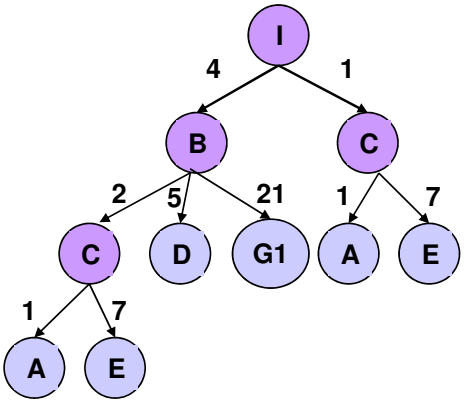
Cola de nodos abiertos: E A G1 D C

Ejemplo: búsqueda primero en anchura

Espacio de estados



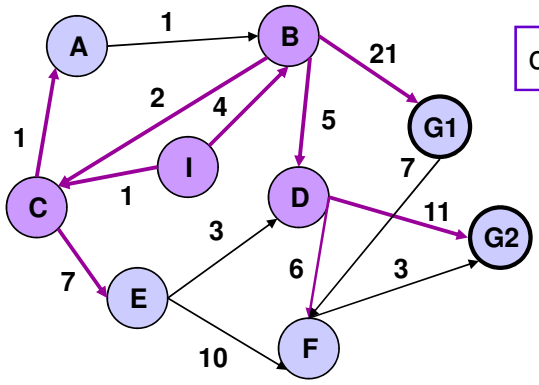
Espacio de búsqueda



Cola de nodos abiertos: E A E A G1 D

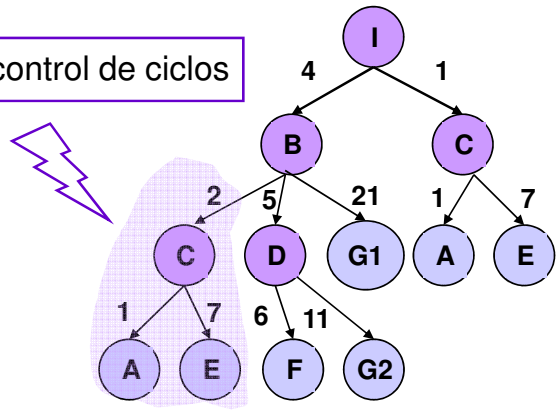
Ejemplo: búsqueda primero en anchura

Espacio de estados



Espacio de búsqueda

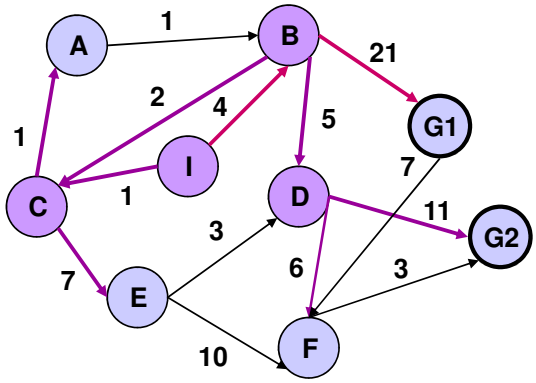
control de ciclos



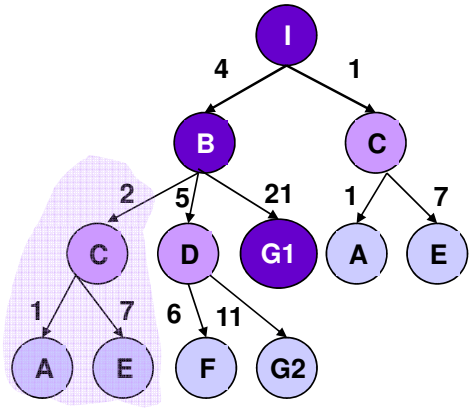
Cola de nodos abiertos: G2 F E A E A G1

Ejemplo: búsqueda primero en anchura

Espacio de estados



Espacio de búsqueda



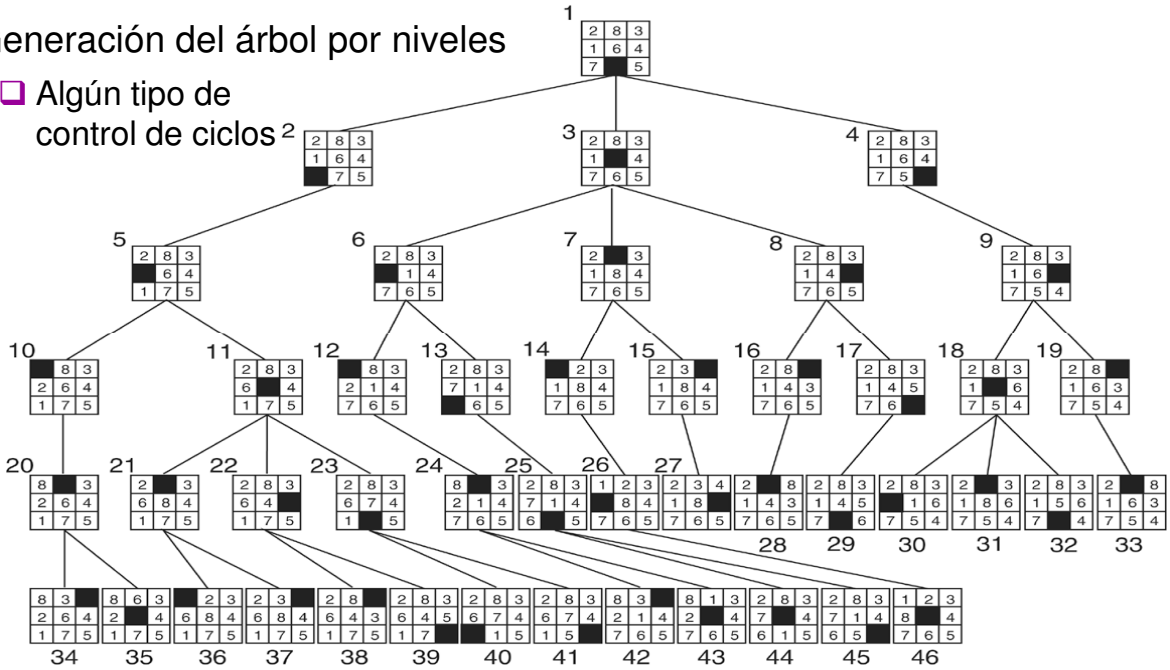
Nodos expandidos (por orden): I B C C D G1

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

Búsqueda primero en anchura (8-puzzle)

- Generación del árbol por niveles
  - Algún tipo de control de ciclos



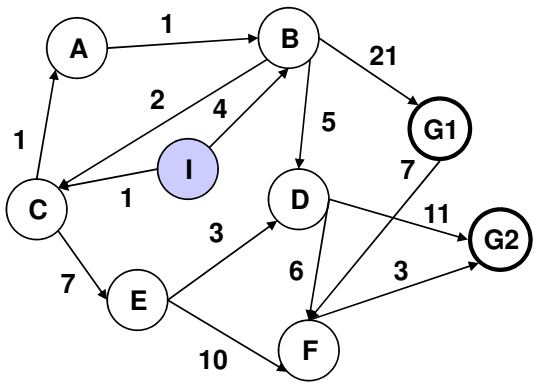
Objetivo

---Búsqueda de coste uniforme (Dijkstra, 1959)---

- Nodos se expanden por orden no decreciente de *coste* del camino (*acumulado*)
  - En cada paso: De los nodos en *abiertos*, se expande el nodo
    - Que tiene el menor coste del camino hasta llegar a él
- *abiertos* se implementación con una cola de prioridad
- Coste del camino frente al número de pasos
  - Si el coste del camino a un nodo es proporcional a su profundidad → todos operadores mismo coste
    - esta búsqueda equivale a primero en anchura
- Propiedades:
  - Completa si no existen caminos infinitos de coste finito
  - Óptima si *coste(sucesor(n)) >= coste(n)* *coste del camino hasta* ***n***
    - Se satisface cuando todos los operadores tienen *coste >= 0*
  - Complejidad en espacio y tiempo equivalente a primero en anchura: $O(r^p)$

Ejemplo: búsqueda de coste uniforme

Espacio de estados

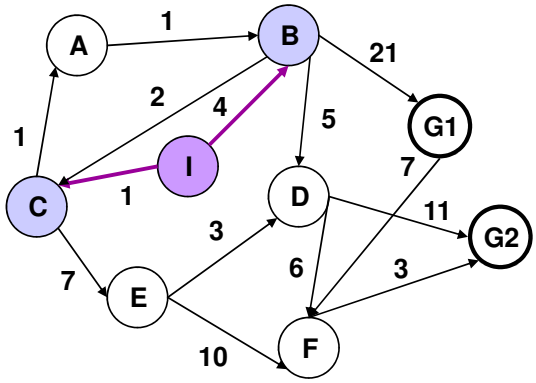


Espacio de búsqueda

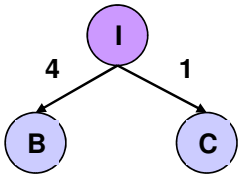
Cola de prioridad de nodos abiertos: I(0)

Ejemplo: búsqueda de coste uniforme

Espacio de estados



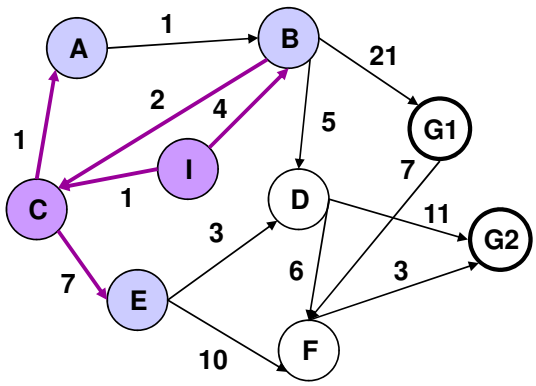
Espacio de búsqueda



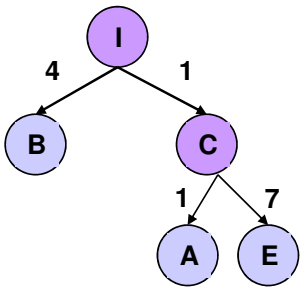
Cola de prioridad de nodos abiertos: B(4) C(1)

Ejemplo: búsqueda de coste uniforme

Espacio de estados



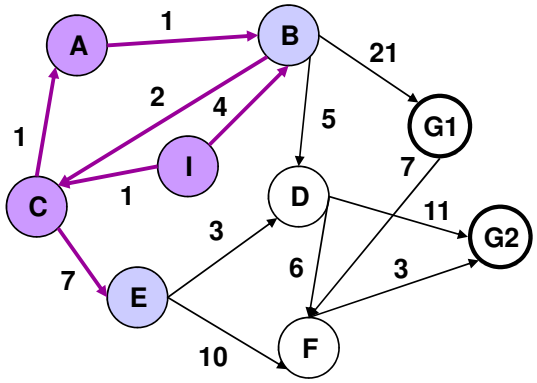
Espacio de búsqueda



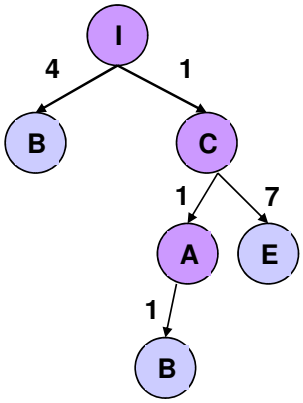
Cola de prioridad de nodos abiertos: E(8) B(4) A(2)

Ejemplo: búsqueda de coste uniforme

Espacio de estados



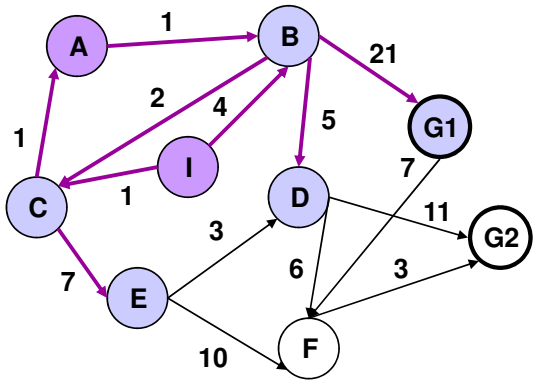
Espacio de búsqueda



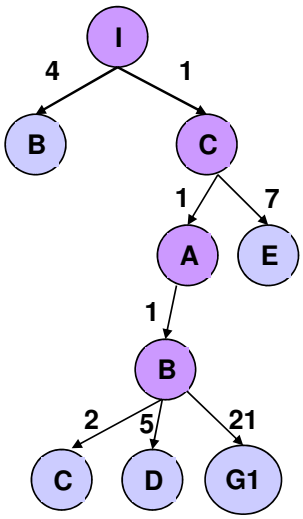
Cola de prioridad de nodos abiertos: E(8) B(4) B(3)

Ejemplo: búsqueda de coste uniforme

Espacio de estados



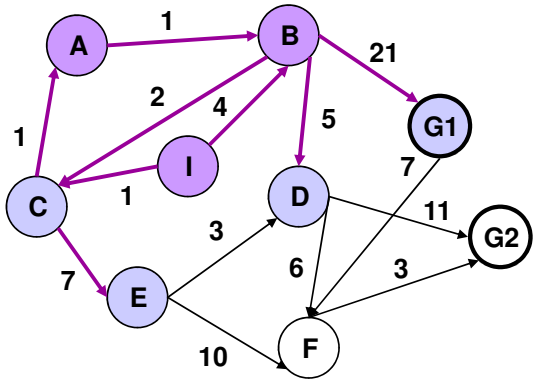
Espacio de búsqueda



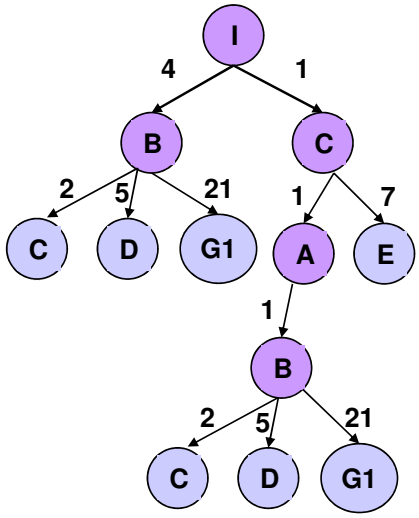
Cola de prioridad de nodos abiertos: G1(24) E(8) D(8) C(5) B(4)

Ejemplo: búsqueda de coste uniforme

Espacio de estados



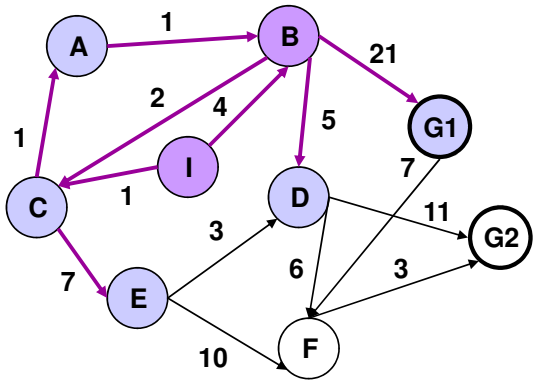
Espacio de búsqueda



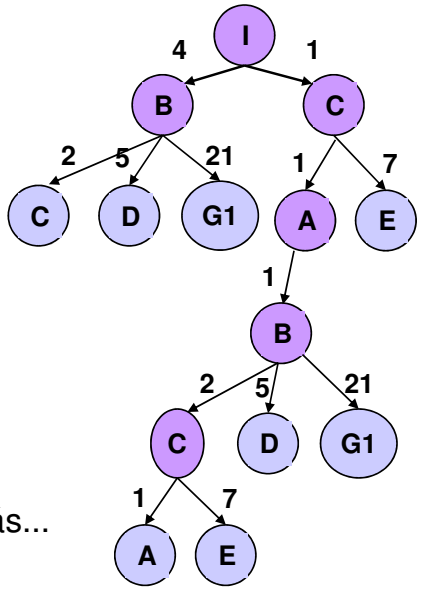
Cola de prioridad de nodos abiertos: G1(25) G1(24) D(9) E(8) D(8) C(6) C(5)

Ejemplo: búsqueda de coste uniforme

Espacio de estados



Espacio de búsqueda

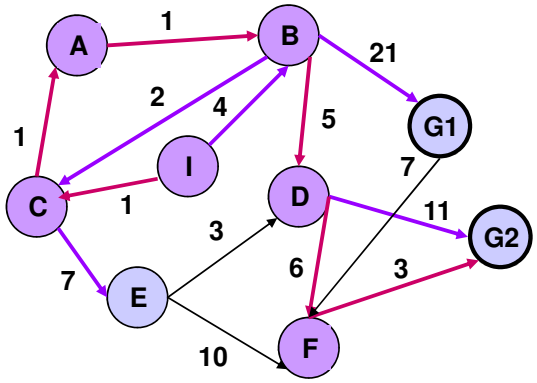


Y así seguiría unos cuantos pasos más...

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

Ejemplo: búsqueda de coste uniforme

Espacio de estados



Búsqueda de coste uniforme

- Completa y óptima *(en este caso!)*
- 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 sólo sobre la rama actual si funciona*
- La mejor solución se encuentra a profundidad 6 **:**

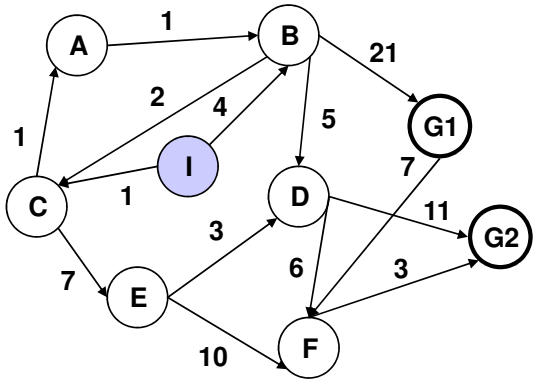
Camino a la solución: I C A B D F G2 Coste: 1+1+1+5+6+3 = 17

--- Búsqueda primero en profundidad ---

- Los nodos se expanden por orden inverso de llegada (sigue una rama):
  - En cada paso:
    - El nodo actual es el último de *abiertos*
    - Si el nodo actual no tiene sucesores y no es objetivo
      - Lo quita de abiertos (se hace backtracking o vuelta atrás)
    - En caso contrario: se expande (y se quita de *abiertos*)
- La estructura *abiertos* se implementa con una pila (o usa recursión)
- Propiedades:
  - No completa: puede meterse en caminos infinitos
  - No óptima: puede haber soluciones mejores por otros caminos; no recomendable si ***m*** es grande
  - Espacio: $O(r*m)$, ***m*** es la máxima prof. del árbol y ***r*** factor de ramificación
    - requisitos modestos: basta el camino actual y los nodos no expandidos
  - Tiempo: $O(r^m)$ *(si hay muchas soluciones, puede ser más rápida que primero en anchura; depende del orden de aplicación de operadores)*

Ejemplo: búsqueda primero en profundidad

Espacio de estados

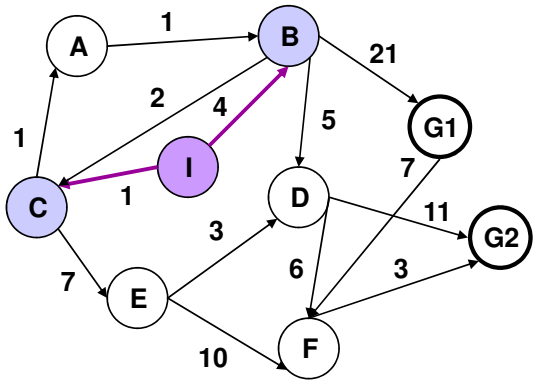


Espacio de búsqueda

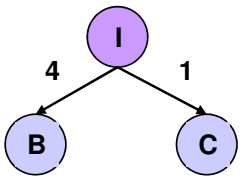
Pila de nodos abiertos: I

Ejemplo: búsqueda primero en profundidad

Espacio de estados



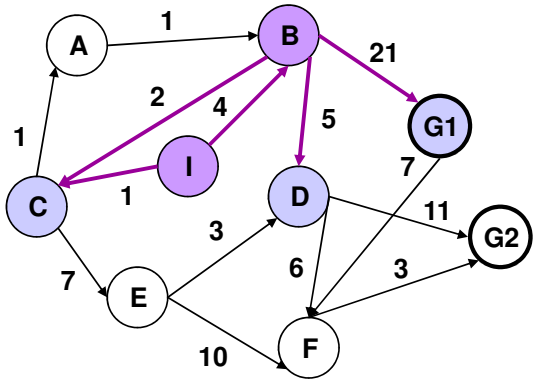
Espacio de búsqueda



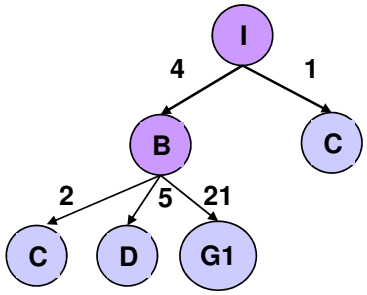
Pila de nodos abiertos: B C

Ejemplo: búsqueda primero en profundidad

Espacio de estados



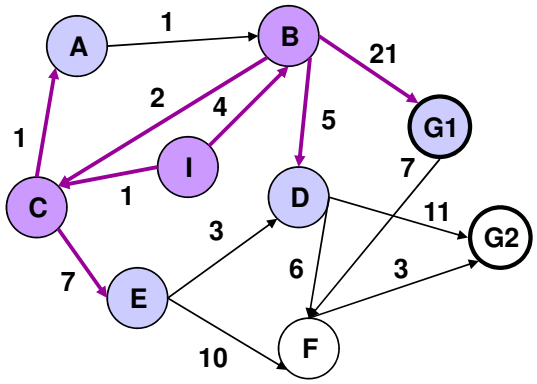
Espacio de búsqueda



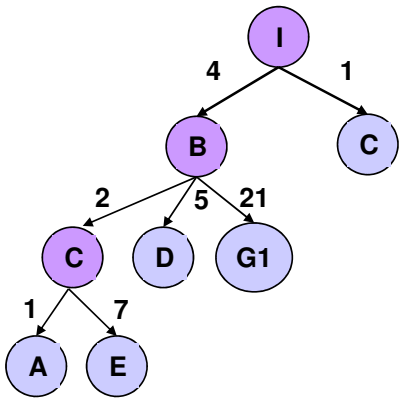
Pila de nodos abiertos: C D G1 C

Ejemplo: búsqueda primero en profundidad

Espacio de estados



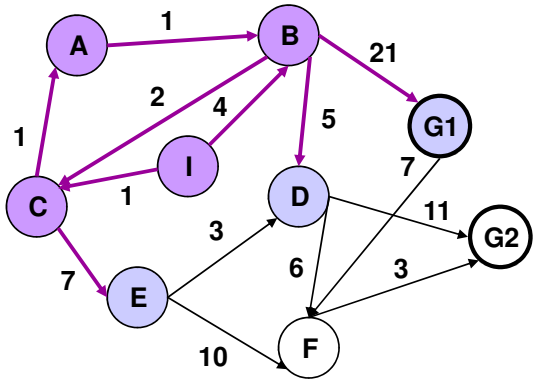
Espacio de búsqueda



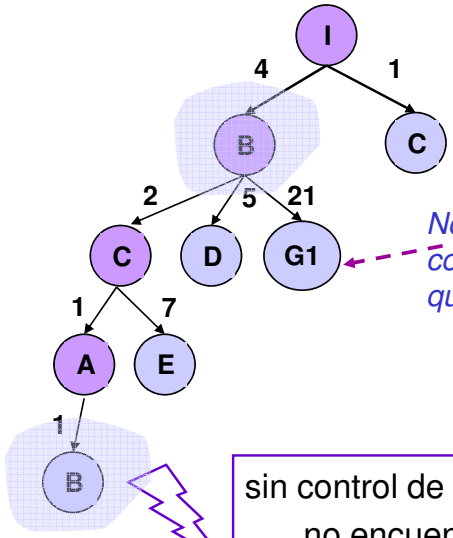
Pila de nodos abiertos: A E D G1 C

Ejemplo: búsqueda primero en profundidad

Espacio de estados



Espacio de búsqueda



Pila de nodos abiertos: B E D G1 C

Nodos expandidos cerrados: I B C A

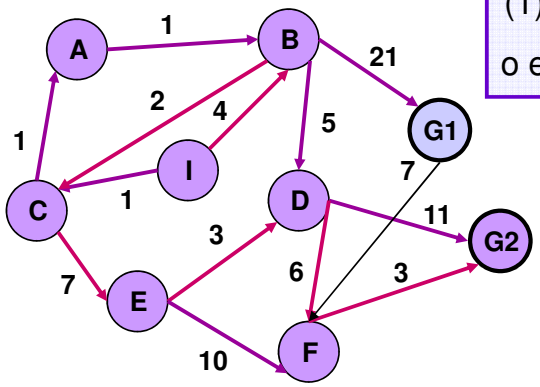
sin control de ciclos
no encuentra solución
(rama infinita)

Control de ciclos: evitar repeticiones de estados

- Cuando el espacio de estados tiene ciclos conviene controlarlos
  - Por eficiencia o por terminación
  - *"Los algoritmos que olvidan su historia están condenados a repetirla"*
- Pero su control empeora la complejidad de los algoritmos utilizados
  1. Mirar los nodos del camino actual es lo más sencillo, lo menos costoso *(cada nodo tiene acceso a su padre)* y lo más seguro
  2. Mirar los nodos de abiertos *(los generados aún pendientes de expandir)*
  3. Mirar los estados ya expandidos:
     - Para ello es necesario tenerlos almacenados (estructura "*cerrados*")
  4. Combinar las 2 últimas comprobaciones
     - Así, cada estado del espacio de estados es examinado, a lo más, una vez
     - Puede suponer una sobrecarga inaceptable o un derroche *(si no es necesario)*
       - Gestión de cerrados *(además, siempre aumenta; nunca se eliminan elementos)*
       - Aplicación de algoritmos de búsqueda en abiertos y cerrados
     - ¡Cuidado! Pueden perderse propiedades de las estrategias

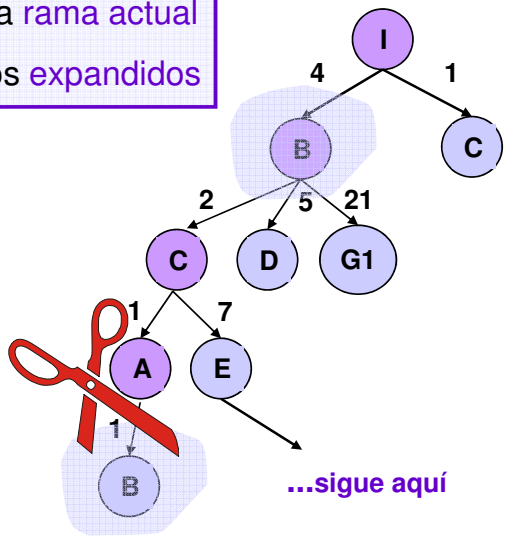
--- Primero en profundidad + control ciclos (1) ---

Espacio de estados



Espacio de búsqueda

(1) Mirar en la rama actual o en los nodos expandidos



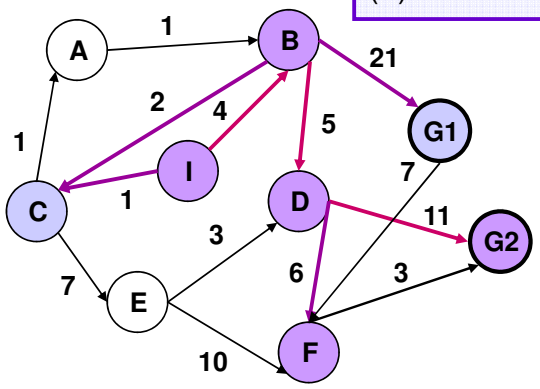
...sigue aquí

Nodos expandidos: 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

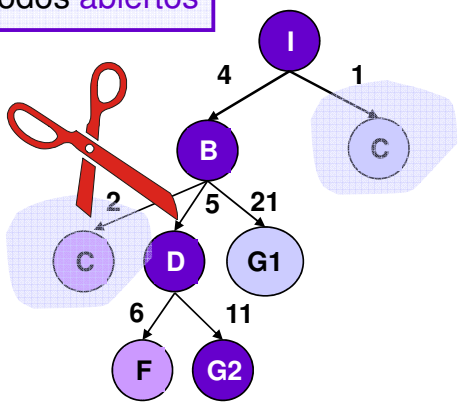
--- Primero en profundidad + control ciclos (2) ---

Espacio de estados



Espacio de búsqueda

(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

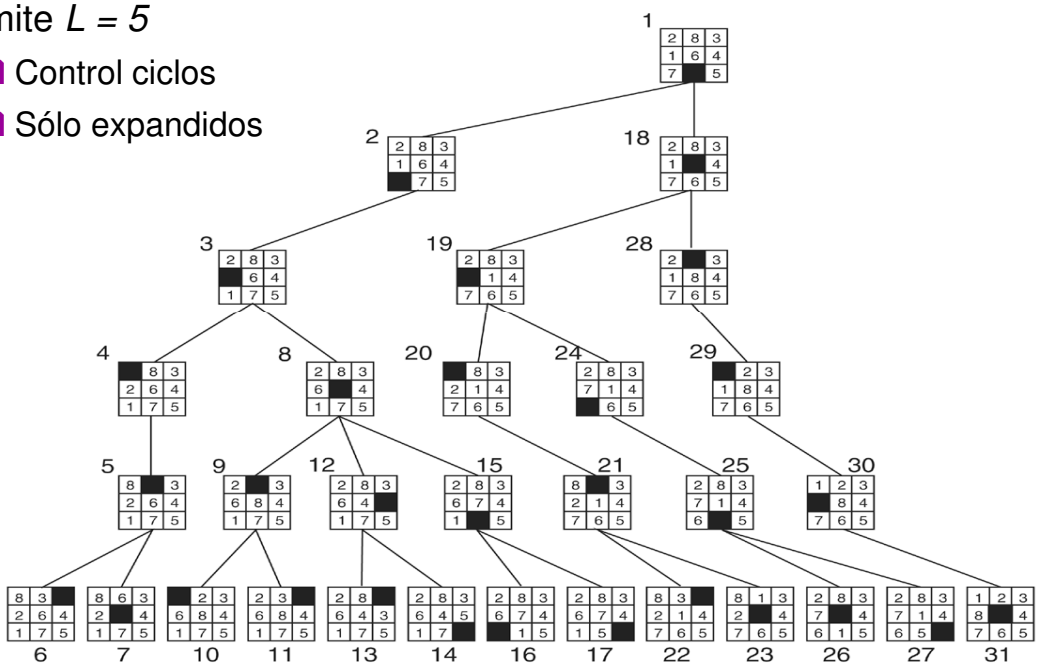
Camino a la solución: I B D G2 Coste: 4+5+11 = 20

--- Búsqueda 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
- Propiedades:
  - *Completa sólo si L >= p* (*p* profundidad mínima de "la solución")
    - 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
  - Tiempo: $O(r^L)$
  - Espacio: $O(r*L)$

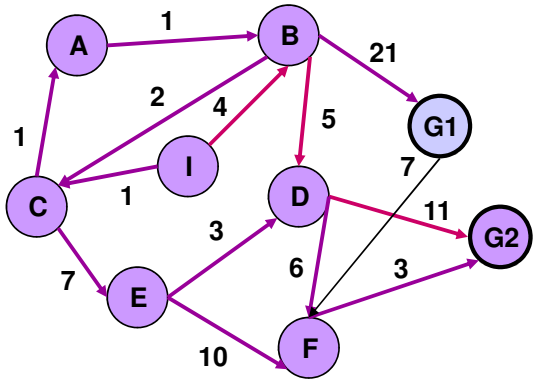
Búsqueda de profundidad limitada

- Límite *L = 5*
  - Control ciclos
  - Sólo expandidos

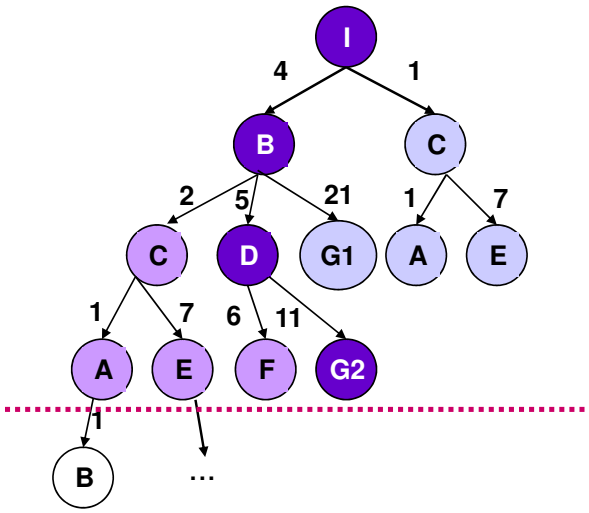


Ejemplo: búsqueda de profundidad limitada *(L = 3)*

Espacio de estados



Espacio de búsqueda



Nodos expandidos: I B C A E D F G2

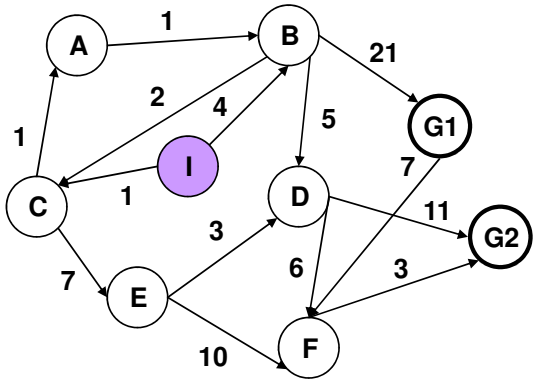
Camino a la solución: I B D G2 Coste: 4+5+11 = 20

--- Búsqueda con profundización 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
- Propiedades:
  - *Óptima y completa (como primero en anchura: la de menor profundidad)* pero ocupando poca memoria (como primero en profundidad)
  - Tiempo: $O(r^p)$ *(algo mejor 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)

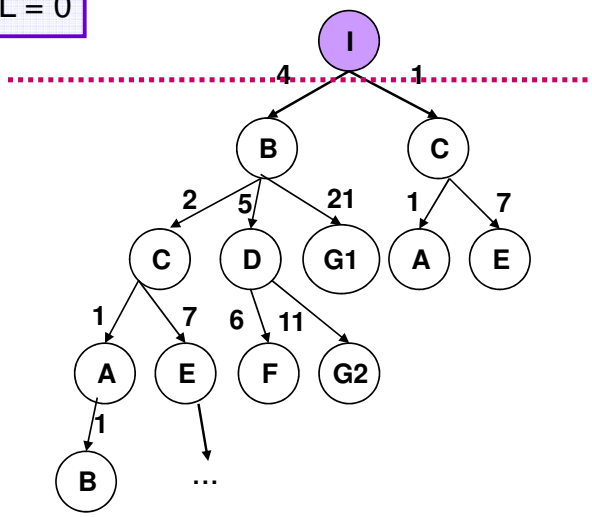
Ejemplo: búsqueda con profundización iterativa

Espacio de estados



Espacio de búsqueda

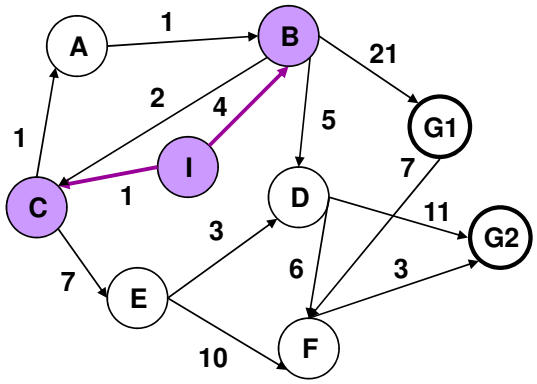
L = 0



Nodos expandidos: I

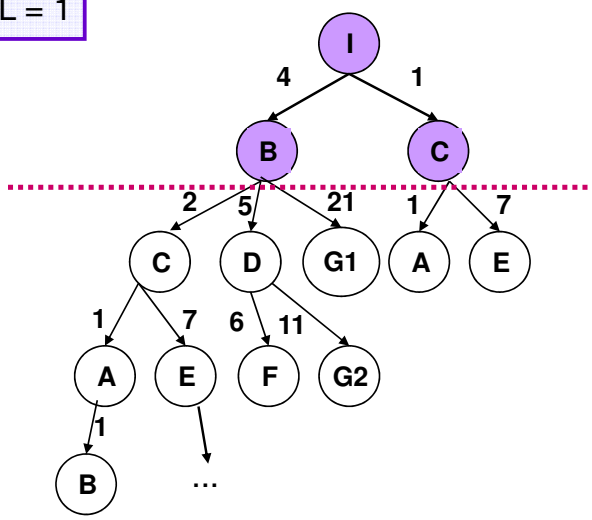
Ejemplo: búsqueda con profundización iterativa

Espacio de estados



Espacio de búsqueda

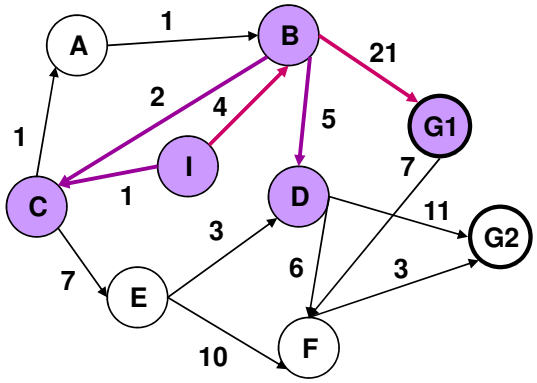
L = 1



Nodos expandidos: I I B C

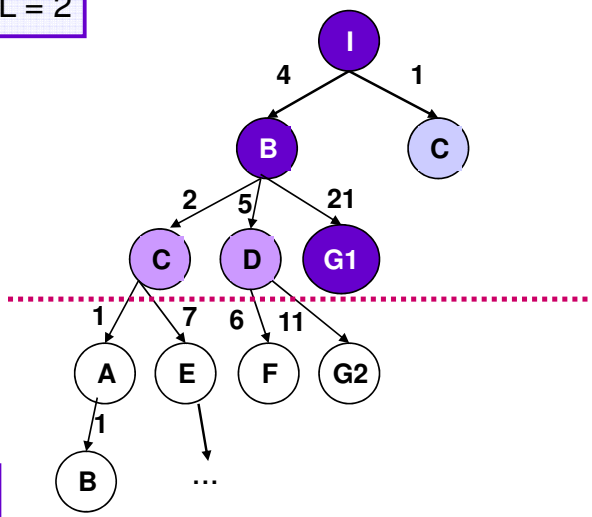
Ejemplo: búsqueda con profundización iterativa

Espacio de estados



Espacio de búsqueda

L = 2



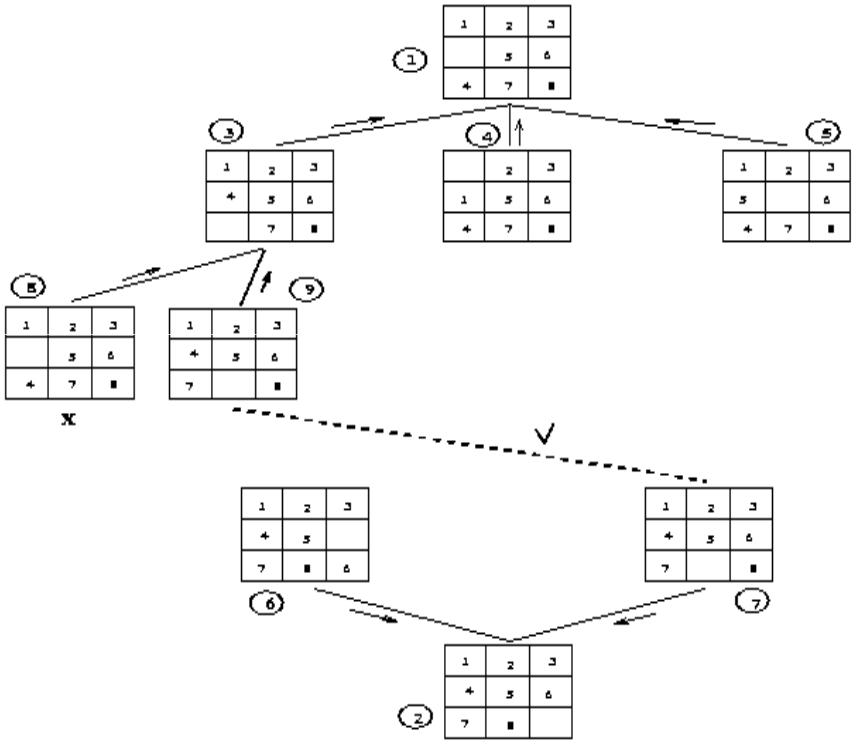
Nodos expandidos: I I B C I B C D G1

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

--- Búsqueda bidireccional (Pohl, 1969) ---

- 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: $r^{p/2} + r^{p/2}$ 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)
- Propiedades:
  - *Óptima y completa si las búsquedas son en anchura y los costes son uniformes*. Otras combinaciones no lo garantizan: ¡no encontrarse!
  - Tiempo: $O(2*r^{p/2}) = O(r^{p/2})$
    - Si la comprobación de la coincidencia puede hacerse en tiempo constante
  - Espacio: $O(r^{p/2})$ *(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 ≈ profundidad	$O(r^p)$	$O(r^p)$
Coste uniforme	Si no hay caminos ∞ de coste finito	Si coste operadores >=0	$O(r^p)$	$O(r^p)$
Primero en profundidad	No	No	$O(r^m)$	$O(r*m)$
Profundidad limitada	Si L >= p	No	$O(r^L)$	$O(r*L)$
Profundización iterativa	Sí	Si coste ≈ profundidad	$O(r^p)$	$O(r*p)$
Bidireccional	Sí (anchura)	Sí	$O(r^{p/2})$	$O(r^{p/2})$

r = factor de ramificación
m = máxima profundidad del árbol

p = profundidad mínima solución
L = límite de profundidad