

Fundamentos de Inteligencia Artificial

Centro Asociado de Melilla

Tutor: Aziz Mulud Tieb

Bloque 2:

Introducción a las Técnicas de Búsqueda

2.1 Concepto de búsqueda en Inteligencia Artificial

2.2 Primero en anchura

2.3 Primero en profundidad

2.4 Coste uniforme

2.5 Búsqueda en profundidad iterativa

2.6 Búsqueda en anchura iterativa

2.7 Búsqueda bidireccional

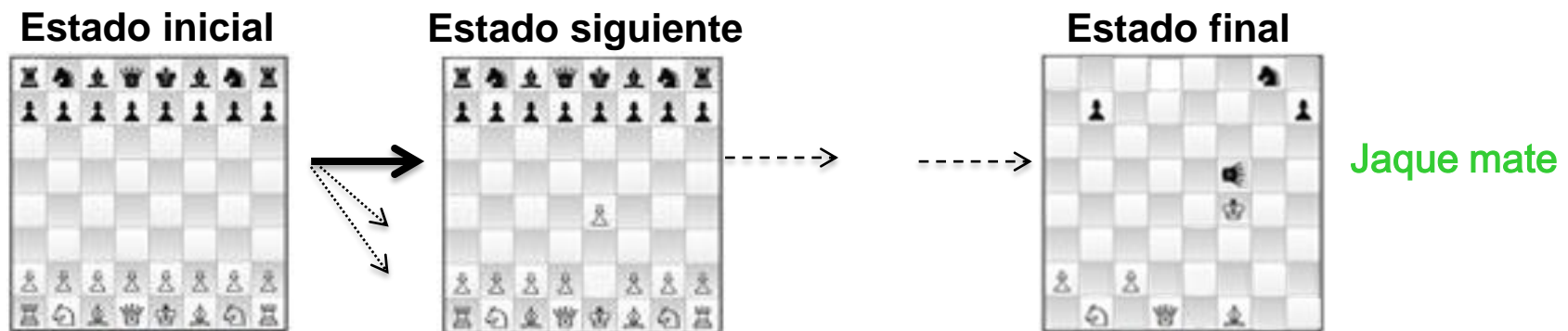
2.1- Concepto de búsqueda en Inteligencia Artificial

¿Qué son las técnicas de búsqueda y cuáles son sus elementos?

- Son **esquemas de representación del conocimiento**,
- Usan diversos algoritmos que permiten resolver ciertos problemas desde el punto de vista de la I.A.

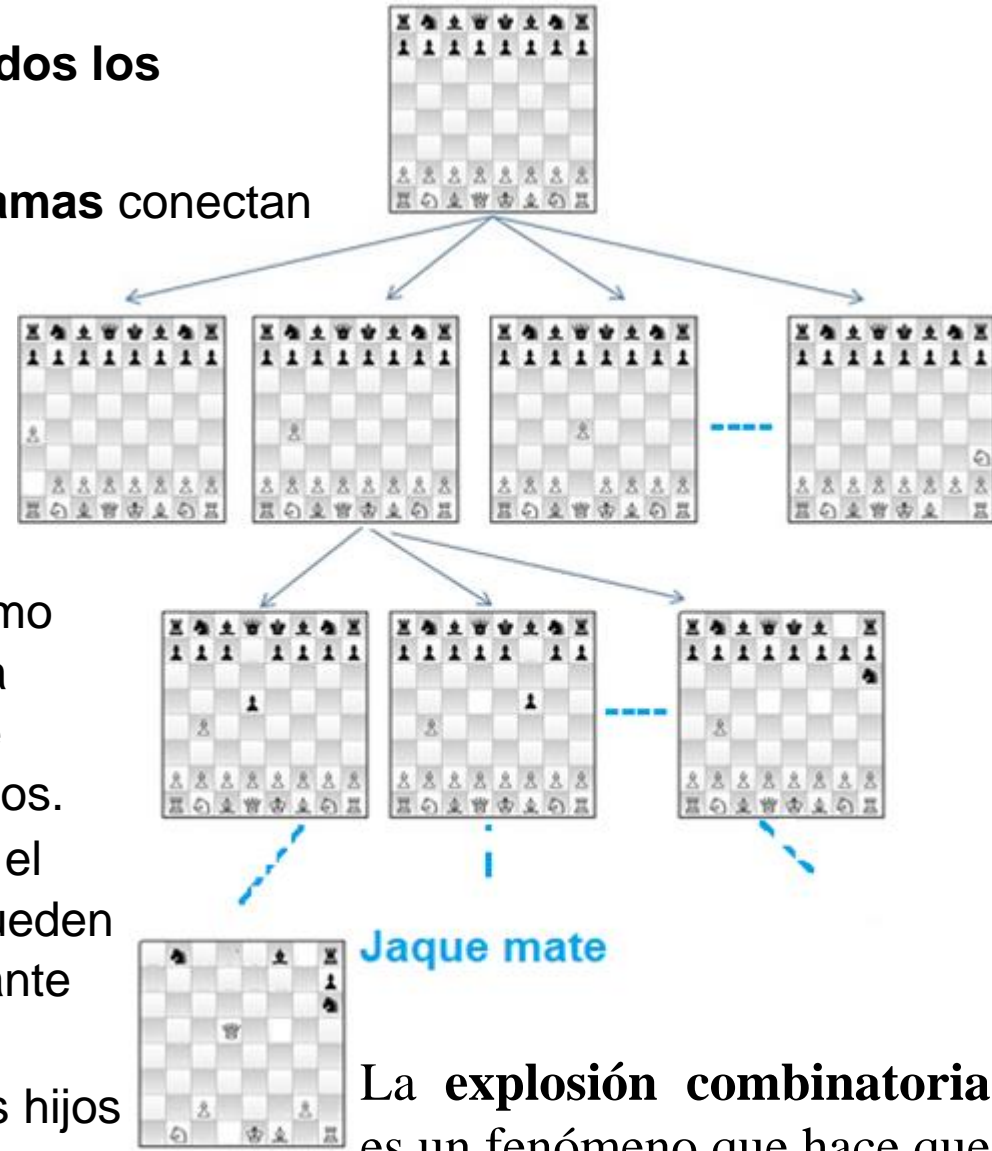
Los **elementos** que integran las técnicas de búsqueda son:

- **Conjunto de estados**: todas las configuraciones posibles en el dominio.
- **Estado inicial**: estado inicial del problema de partida.
- **Estados finales**: las soluciones del problema.
- **Operadores**: se aplican para pasar de un estado a otro.
- **Solucionador**: mecanismo que nos permite evolucionar de un estado a otro mediante un algoritmo aplicando



Árbol de búsqueda

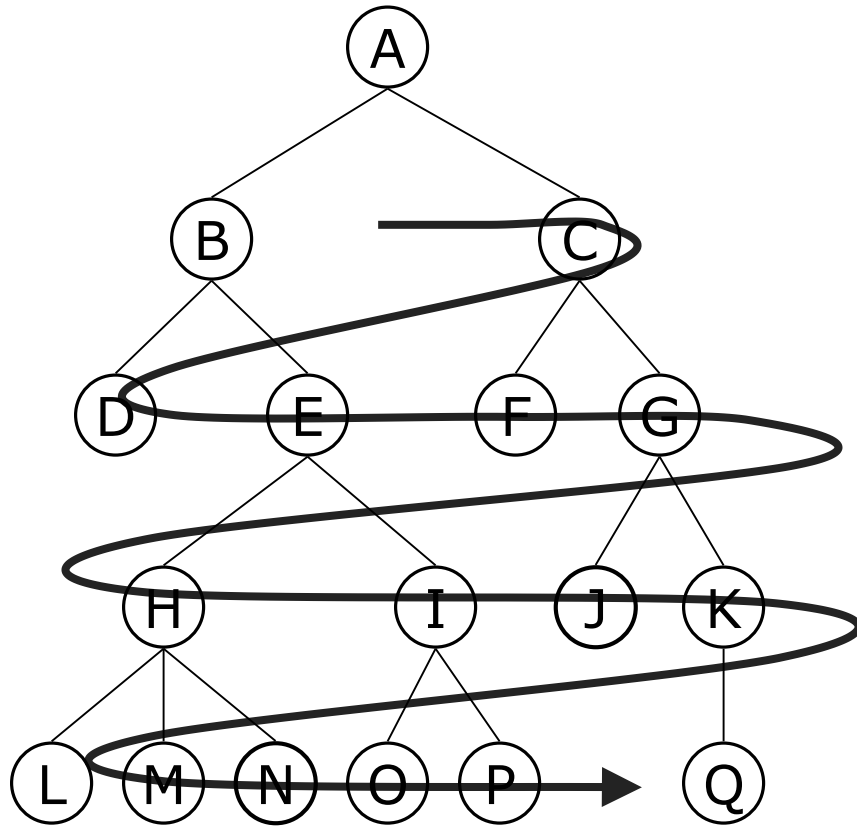
- Es una representación que considera **todos los estados (trayectorias) posibles**.
- Los **nodos** representan estados, y las **ramas** conectan estados
- Existen dos tipo de Búsqueda
 - Búsqueda ciega:
 - Búsqueda Heurística



- **Búsqueda ciega:** conocida también como **búsqueda sin información**, realiza una **exploración exhaustiva** del espacio de estados. Todos los nodos son examinados.
- **Espacio de estados del problema:** Es el conjunto de todos los estados que se pueden alcanzar a partir del estado inicial mediante cualquier secuencia de acciones.
- **Expandir un nodo:** obtener los posibles hijos de un nodo.
- **Ruta:** es cualquier secuencia de acciones que me permite pasar de un estado a otro.

La **explosión combinatoria** es un fenómeno que hace que el problema no se pueda abordar computacionalmente.

2.2 Primero en anchura



- ❑ Recorre los nodos en orden de su distancia desde el origen
- ❑ Visita a todos los nodos (el ancho) de un nivel antes de seguir con el siguiente
- ❑ Siempre encuentra la distancia mínima entre un nodo y el origen
- ❑ Es un método **completo** ya que siempre encuentra la solución en caso de que exista.
- ❑ Es un método **admisible** en el caso de que todas las reglas tengan coste uniforme ya que se encuentra la solución óptima

Primero en anchura: Algoritmo

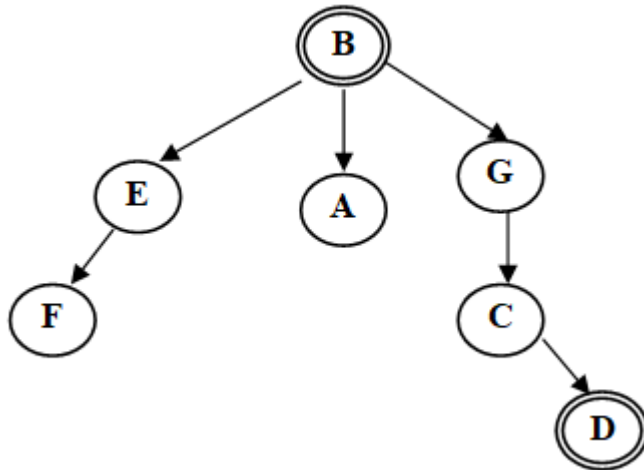
Algoritmo de búsqueda sin información en árboles

```
ABIERTA = (inicial);  
mientras NoVacía(ABIERTA) hacer  
     $n = \text{ExtraePrimero}(\text{ABIERTA});$   
    si EsObjetivo( $n$ ) entonces  
        devolver Camino(inicial,  $n$ ); {leyéndolo en la TABLA_A}  
    fin si  
     $S = \text{Sucesores}(n);$   
    para cada  $q$  de  $S$  hacer  
        pone  $q$  en la TABLA_A con  
             $\text{Anterior}(q) = n,$   
             $\text{Coste}(\text{inicial}, q) = \text{Coste}(\text{inicial}, n) + \text{Coste}(n, q);$   
        inserta  $q$  en ABIERTA;  
    fin para  
fin mientras  
devolver “no encontrado”;
```

- La búsqueda en anchura se obtiene insertando los sucesores siempre al final de ABIERTA, con lo que ABIERTA se trata realmente como una COLA.
- Tanto el tiempo de ejecución como el espacio de memoria necesario, crecen de forma exponencial con el tamaño del problema.


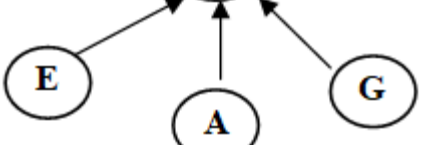
Primero en anchura: ejemplo


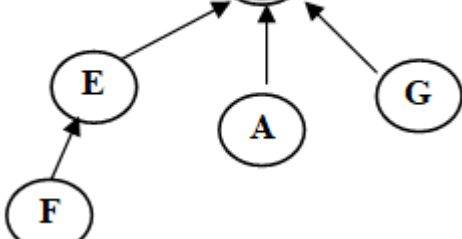
Espacio de búsqueda



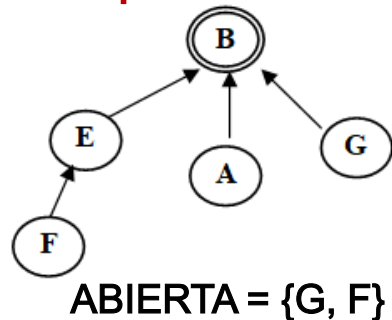
Árbol parcialmente desarrollado

Paso 0  ABIERTA = {B}

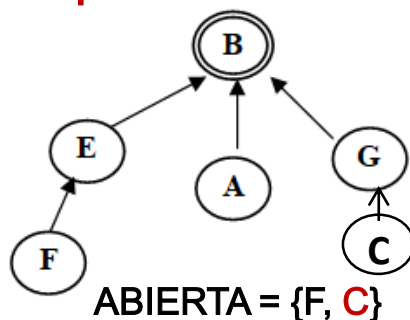
Paso 1  **Expandimos B**
 ABIERTA = {E, A, G}

Paso 2  **Expandimos E**
 ABIERTA = {A, G, **F**}

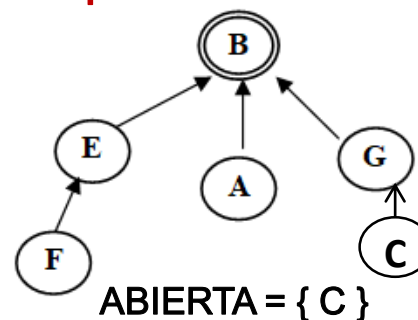
Paso 3:
Expandimos A



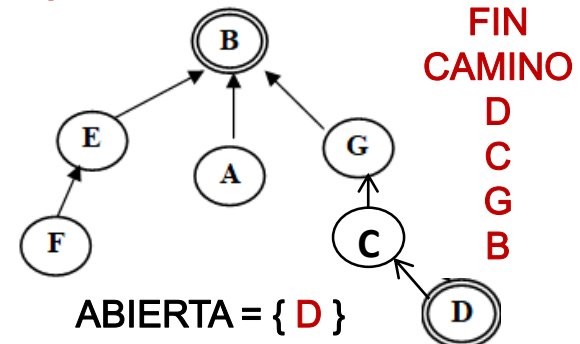
Paso 4:
Expandimos G



Paso 5:
Expandimos F



Paso 6:
Expandimos C



Paso 7
ABIERTA = { }

FIN CAMINO
D C G B

Primero en anchura: Complejidad

➤ Definiciones:

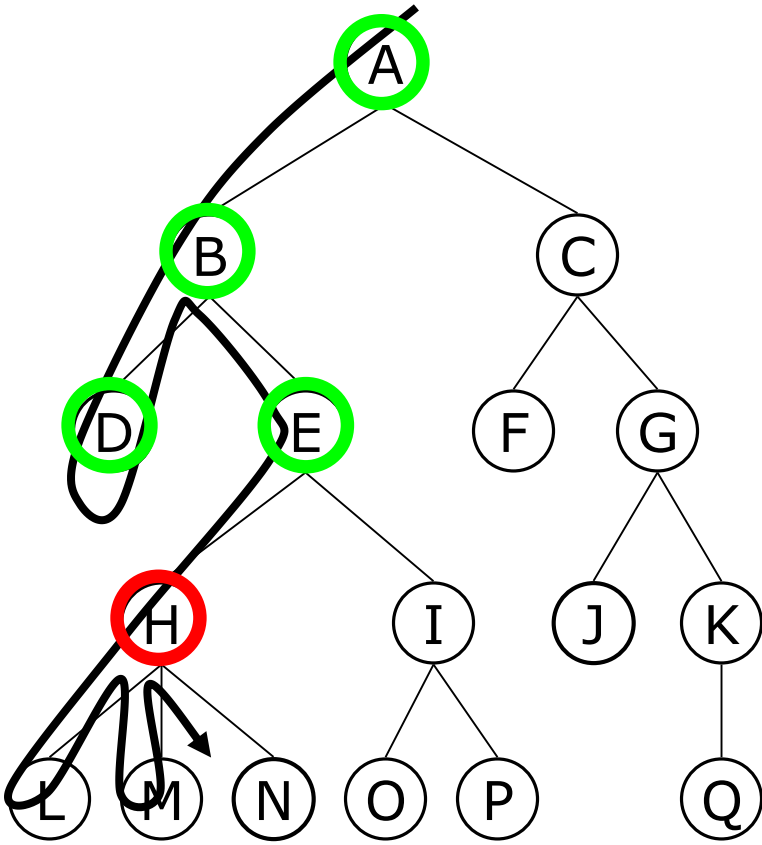
- **Factor de Ramificación (b):** número medio de hijos de un nodo del espacio de búsqueda
- **Profundidad de la solución (d):** número de arcos desde el nodo inicial hasta el nodo meta

➤ **Complejidad temporal:** tiempo de ejecución empleado en obtener la solución. Depende del número de nodos generados. $O(b^{d+1})$

➤ **Complejidad espacial:** recursos (memoria) necesarios para obtener la solución. También depende del número de nodos generados ya que hay que almacenar todas las ramas exploradas. $O(b^{d+1})$

➤ El peor de los casos se da cuando la meta está en el último nodo del último nivel d

2.3 Primero en profundidad

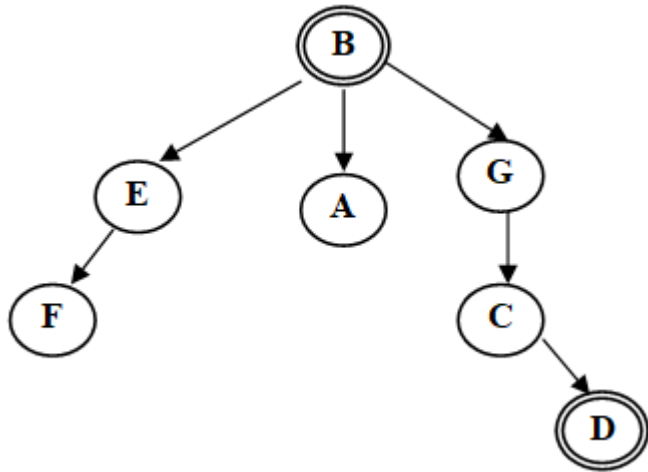


- Recorre todos los nodos de una rama antes de seguir con la siguiente
- Visita a algunos nodos muy lejanos (profundos) del origen antes de otros cercanos
- La implementación más fácil es recursiva
- Para no visitar a un mismo nodo varias veces, hay que *marcar* los nodos ya visitados
- En este caso los nodos se insertan al principio de ABIERTA, con lo cual esta estructura se trata como una **pila**.

- **No es admisible ni completo.** Se debe a que la búsqueda se puede derivar hacia una rama infinita, con lo que el **algoritmo no termina**.
- Para evitar este inconveniente, se suele **establecer una profundidad límite** a partir de la cual se detiene la búsqueda. En este caso el algoritmo termina, pero tampoco hay garantía de que sea completo ya que la solución puede encontrarse más allá de la profundidad límite elegida.


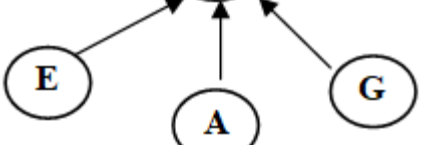
Primero en profundidad : ejemplo


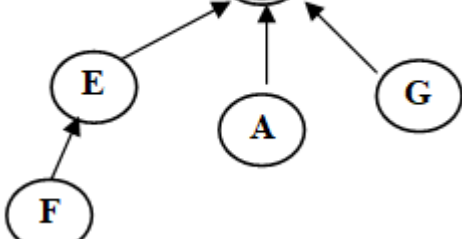
Espacio de búsqueda

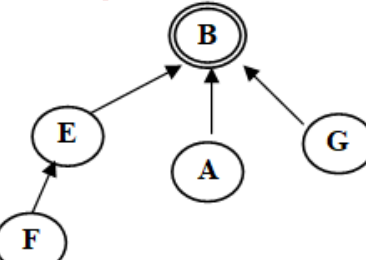


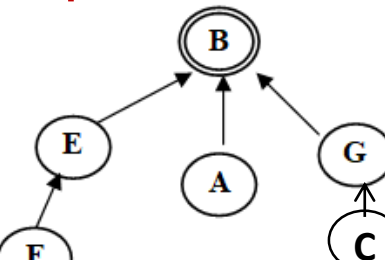
Árbol parcialmente desarrollado

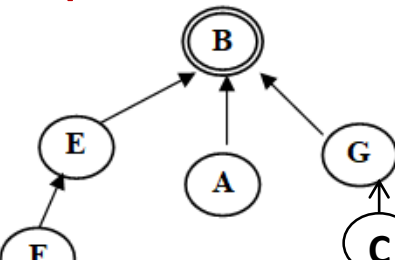
Paso 0  ABIERTA = {B}

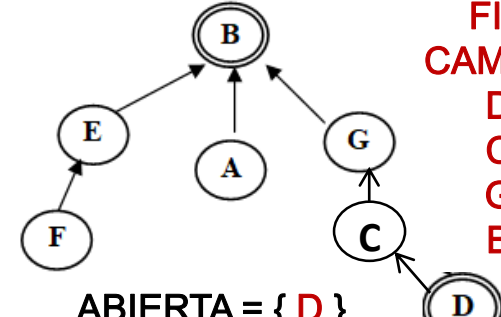
Paso 1  **Expandimos B**
 ABIERTA = {E, A, G}

Paso 2  **Expandimos E**
 ABIERTA = {**F**, A, G}

Paso 3:
Expandimos F

ABIERTA = {A, G}

Paso 4:
Expandimos A

ABIERTA = {G}

Paso 5:
Expandimos G

ABIERTA = {**C**}

Paso 6:
Expandimos C

ABIERTA = {**D**}

Paso 7
ABIERTA = { }
FIN CAMINO
D C G B

2.4 Coste uniforme

- ABIERTA no actúa ni como pila ni como cola.
- Los nodos se insertan de forma ordenada teniendo en cuenta el coste desde el nodo inicial a cada uno de los nodos, se obtiene la estrategia del coste uniforme.
- Se trata de una estrategia similar a la búsqueda en anchura: de hecho, si el coste de todas las reglas es el mismo, las dos estrategias son equivalentes.
- El coste computacional en tiempo y espacio es el mismo que en la búsqueda en anchura, pero la solución que calcula la estrategia del coste uniforme es la de menor coste al inicial, con lo que el algoritmo es admisible.
- **Es completa:** Si existe una solución la encontrará.
- **Es óptima:** Encuentra primero la ruta de menor costo, siempre que dicho costo no disminuya al aumentar la profundidad.
- **Complejidad** (espacial y temporal): $O(b^d)$, dónde **b** es el factor de ramificación y **d** la profundidad de la solución.

2.5 Búsqueda en profundidad iterativa

- Una forma de resolver el problema de la profundidad límite de la búsqueda en profundidad podría ser considerar de forma iterativa sucesivos valores de este parámetro, y para cada uno de ellos realizar una búsqueda completa.
- Esta es la estrategia denominada búsqueda en profundidad iterativa.
- El principal inconveniente de este tipo de búsqueda es que para cada uno de los valores de la **profundidad límite** hay que visitar de nuevo todos los visitados en la iteración anterior.
- Al igual que en la búsqueda en profundidad convencional, si la profundidad máxima alcanzada es d , el espacio de memoria es proporcional a d , y el tiempo de búsqueda es exponencial en d .

Algoritmo de búsqueda primero en profundidad iterativa en árboles

```
1: ProfundidadLimite = 1;  
2: Solucion = BusquedaPrimeroenProfundidad;  
3: mientras Solucion = "no encontrado" hacer  
4:   ProfundidadLimite = ProfundidadLimite + 1;  
5:   Solucion = BusquedaPrimeroenProfundidad;  
6: fin mientras  
7: devolver Solucion;
```

2.6 Búsqueda en anchura iterativa

```
AnchuraLimite = 0;
mientras se pueda aumentar la AnchuraLimite hacer
    AnchuraLimite = AnchuraLimite + 1;
    ABIERTA = (inicial);
    mientras NoVacia(ABIERTA) hacer
        n = ExtraePrimero(ABIERTA);
        si EsObjetivo(n) entonces
            devolver Camino(inicial, n);
        fin si
        S = Sucesores(n, AnchuraLimite);
        para cada q de S hacer
            poner q en la TABLA_A con
                Anterior(q) = n,
                Coste(inicial, q) = Coste(inicial, n) + Coste(n, q);
            insertar q al final de ABIERTA;
        fin para
    fin mientras
fin mientras
devolver “no encontrado”;
```

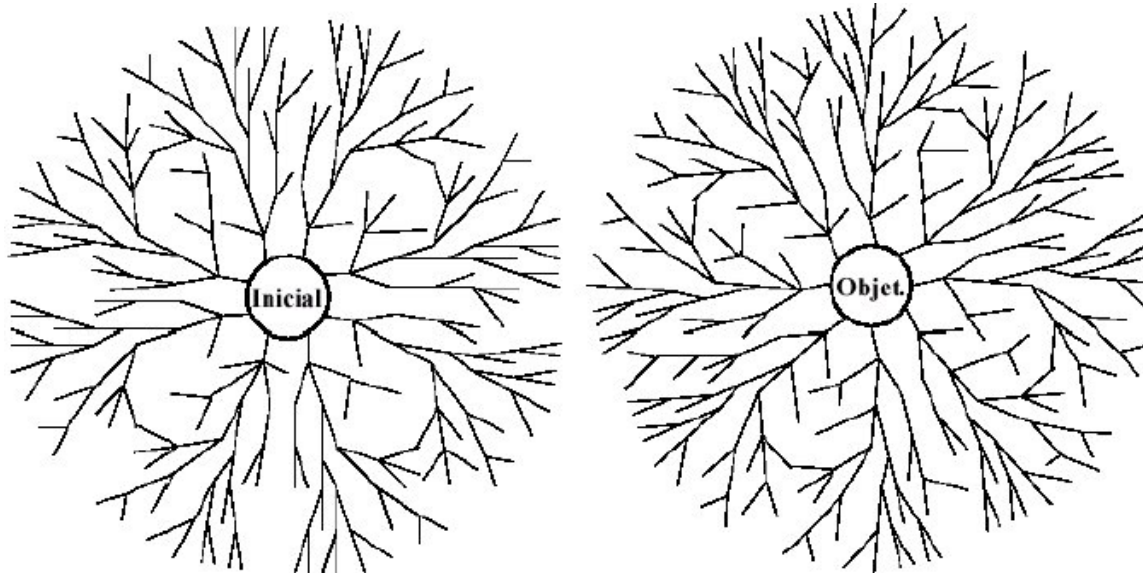
AnchuraLimite permite controlar el número máximo de sucesores de cada estado.

Sucesores(*n*, *AnchuraLimite*) selecciona el número de sucesores que se consideran en cada iteración, de acuerdo con el parámetro *AnchuraLimite*.

- La iteración puede continuar mientras esta operación haya dejado fuera a alguno de los sucesores de algún nodo expandido.
- Este algoritmo requiere un espacio y un tiempo que son exponenciales en la profundidad del árbol de búsqueda.
- La búsqueda primero en anchura iterativa puede encontrar una solución que no sea la más próxima al estado inicial.

2.6 Búsqueda bidireccional

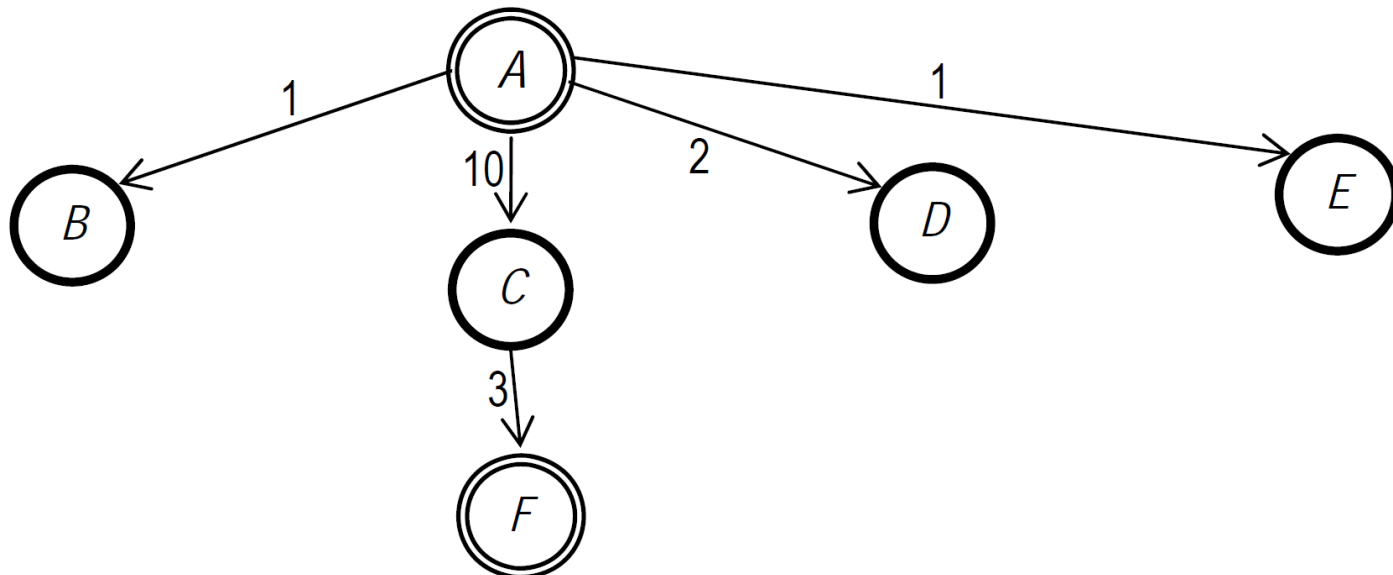
- La idea de la búsqueda bidireccional consiste en **buscar simultáneamente en las dos direcciones**, es decir, hacia delante desde el inicial a los objetivos y hacia atrás desde los objetivos al inicial.
- La búsqueda se detiene cuando las correspondiente fronteras, es decir las dos listas ABIERTA, tienen algún nodo en común.
- De este modo si hay una solución a profundidad d , entonces cada uno de los dos algoritmos tiene que buscar hasta una profundidad $d/2$, con lo que idealmente el tiempo de ejecución será exponencial en $d/2$ en lugar de serlo en d . Lo que supone una mejora realmente importante.



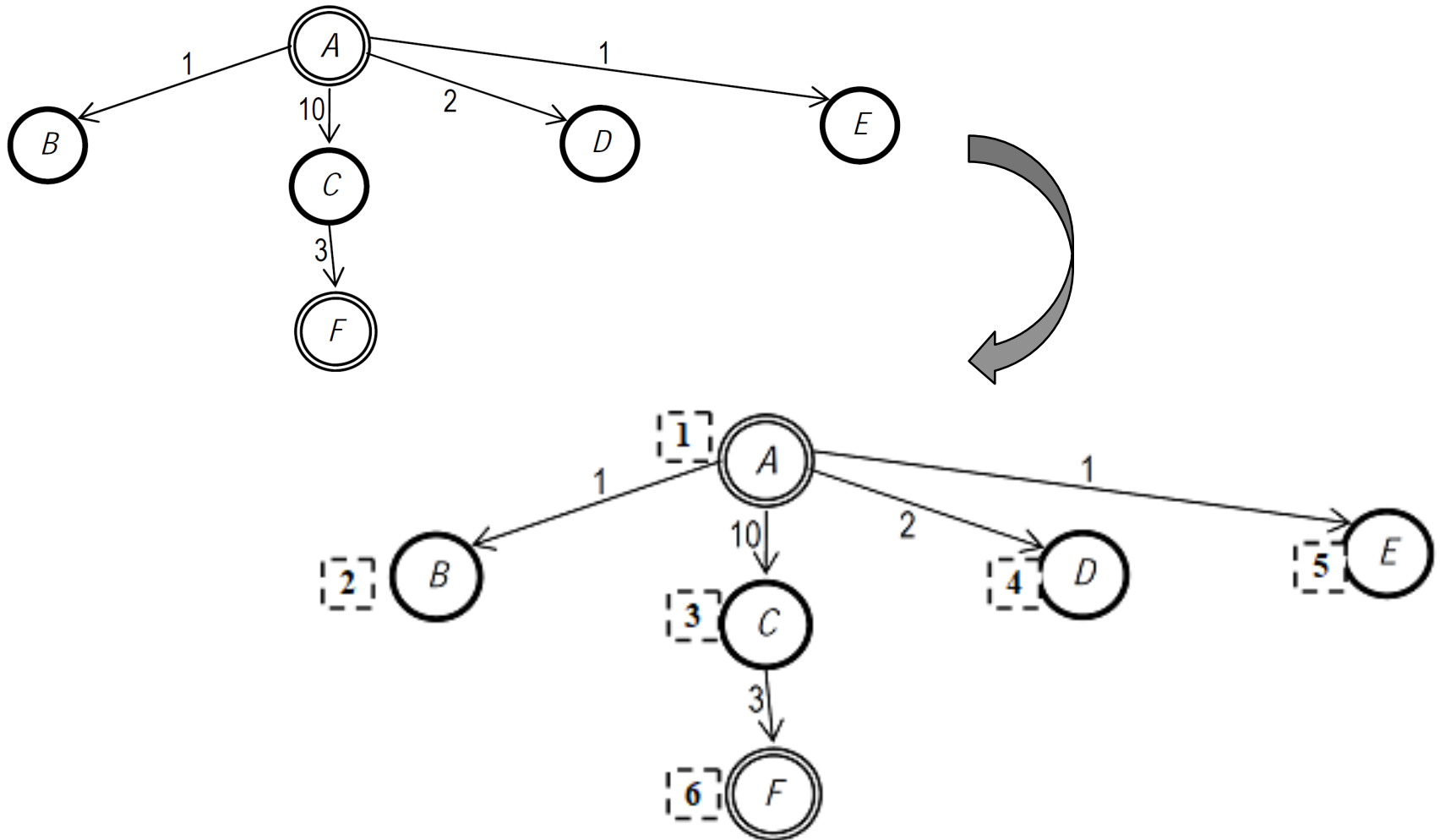
EJERCICIOS

Realice una búsqueda en el siguiente árbol usando las siguientes técnicas de búsqueda:

1. Búsqueda Primero en Anchura (de izquierda a derecha)
2. Búsqueda Primero en Profundidad (de derecha a izquierda)
3. Búsqueda de Coste Uniforme
4. Búsqueda en Anchura Iterativa (de derecha a izquierda)
5. Búsqueda en Profundidad Iterativa (de izquierda a derecha)

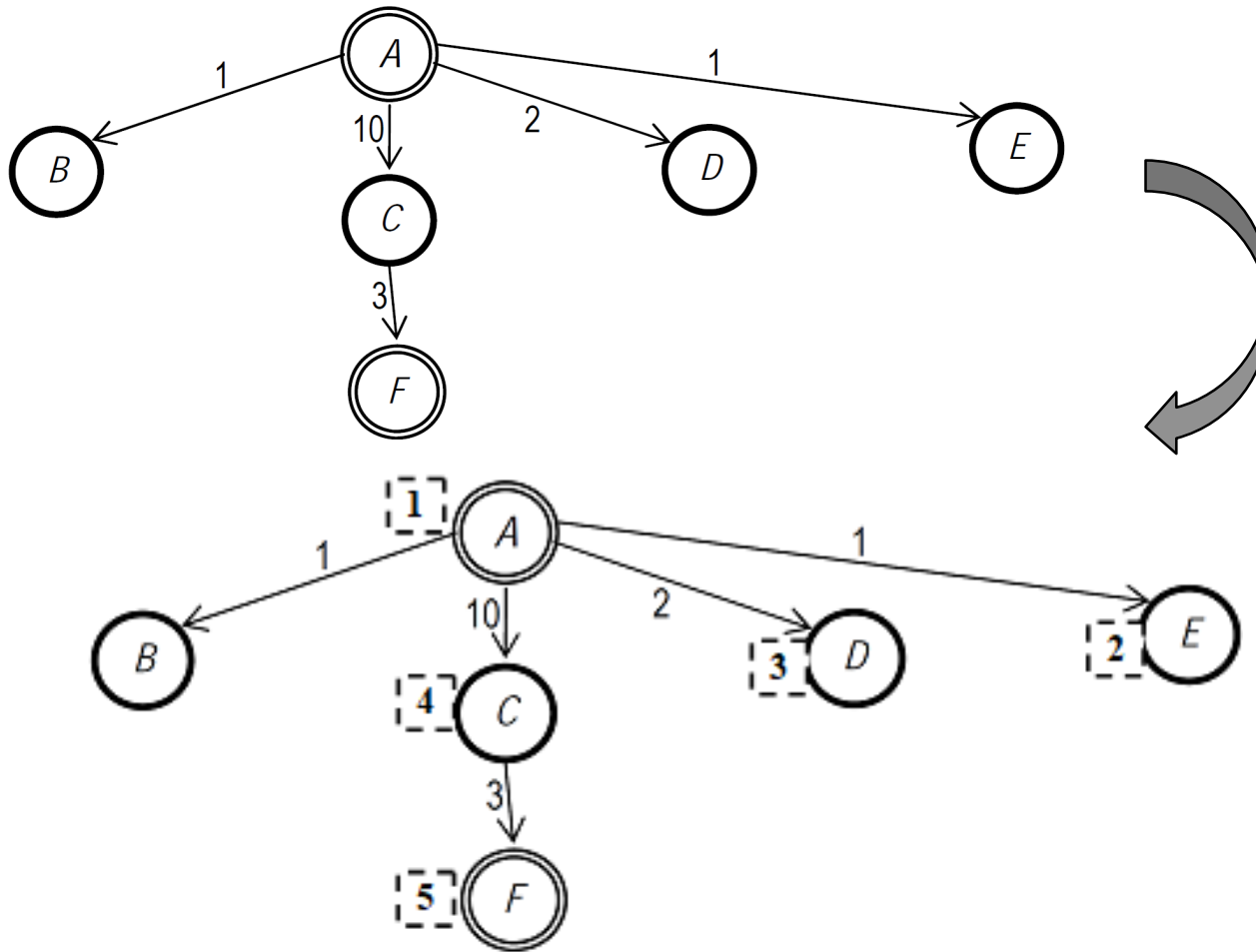


Primero en Anchura (de izquierda a derecha)



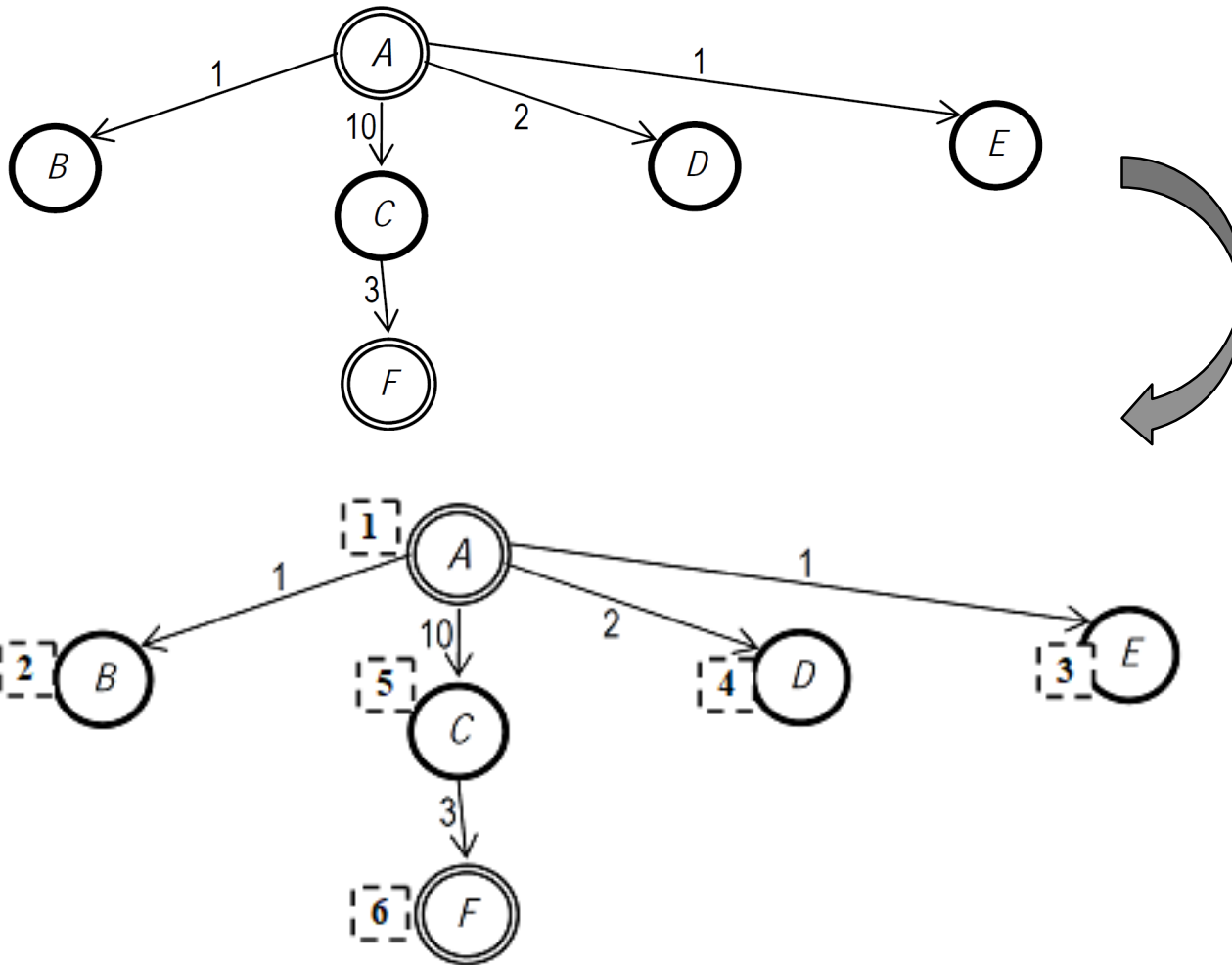
- El orden en el que se visitan los nodos en el caso de la búsqueda en anchura es **A, B, C, D, E, F**.
- Se observa que en este caso se han explorado todos los nodos del árbol para llegar a la meta.

primero en profundidad (de derecha a izquierda)



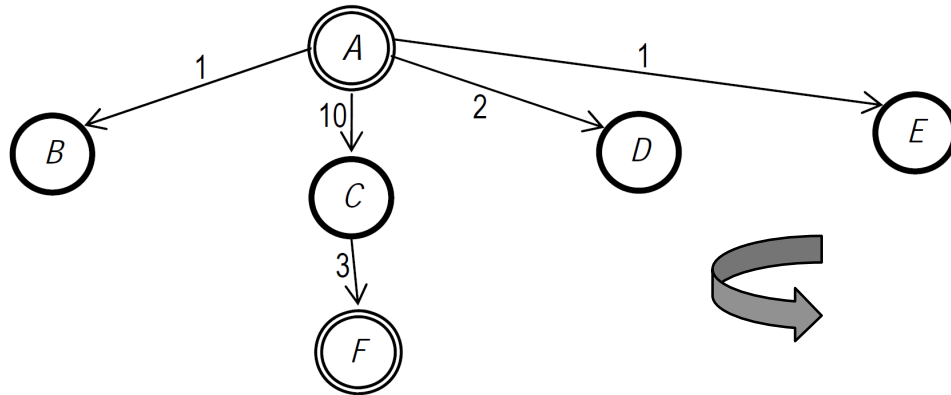
- El orden en el que se visitan los nodos en el caso de la búsqueda en profundidad es: **A, E, D, C, F**.
- Se observa que en este caso se ha explorado un nodo menos que en el caso anterior para llegar a la meta.

Búsqueda de coste uniforme

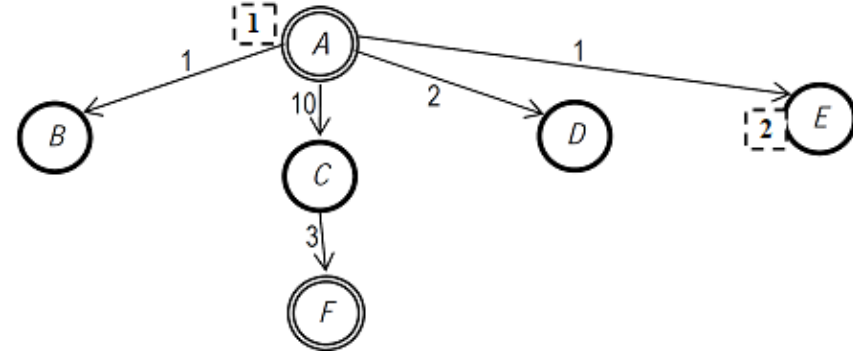


- El orden en el que se visitan los nodos en el caso de la búsqueda de coste uniforme es: **A, B, E, D, C, F**.
- Se observa que en este caso se han explorado todos los nodos del árbol para llegar a la meta.

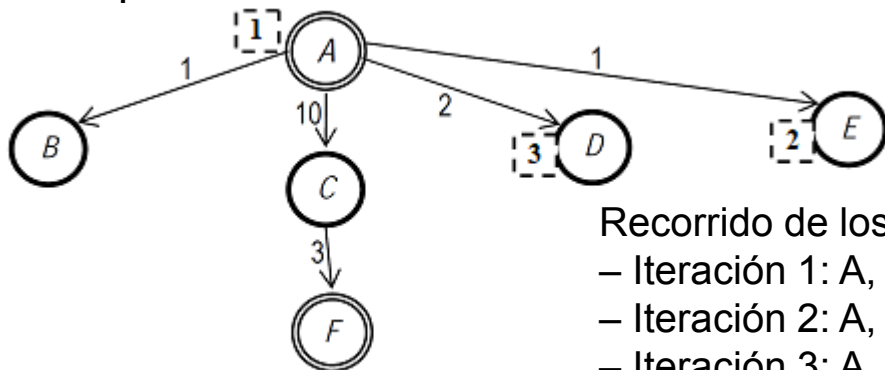
Búsqueda en anchura iterativa (de derecha a izquierda)



ITERACIÓN 1: Primera iteración de la búsqueda en anchura iterativa de derecha a izquierda, en la que como máximo **un hijo** es generado en cada expansión de un nodo padre.



ITERACIÓN 2: Segunda iteración de la búsqueda en anchura iterativa de derecha a izquierda, en la que como máximo **dos hijos** son generados en cada expansión de un nodo padre.



Recorrido de los nodos:

- Iteración 1: A, E
- Iteración 2: A, E, D
- Iteración 3: A, E, D, C, F

ITERACIÓN 3: Tercera y última iteración de la búsqueda en anchura iterativa de derecha a izquierda, en la que como máximo **tres hijos** son generados en cada expansión de un nodo padre.

