Fundamentos de inteligencia artificial

Tema 02 - Introducción a las técnicas de búsqueda

2.1 Introducción

Los problemas de búsqueda reúnen ciertas características y componentes:

- El **dominio** o **espacio de búsqueda**: operadores y estados, grafo dirigido simple.
- Asocia conjunto de **estados** a situaciones del **dominio.**
- Existen **estados iniciales** desde donde comienza el proceso de búsqueda
- Hay **operadores**, que aplicados a un estado, producen otro
- Hay al menos un estado **meta** o **solución**.
- Hay una **estrategia de control**, que determina el orden en que se exploran los nodos.

2.2 Métodos de búsqueda sin información

Realizan un recorrido de forma sistemática, sin tener en cuenta la información del dominio.
Usamos una lista llamada **ABIERTA** (con los estados candidatos a ser desarrollados en cada iteración y ordenados siguiendo cierto criterio) con los nodos generados y no expandidos.
Mantiene los nodos por los que tengo que pasar.
Se calculan los sucesores aplicando todos los operadores posibles.
La función **Sucesores** (para un estado, devuelve una lista de todos sus sucesores).
Usamos una estructura **TABLA_A** (conjunto ordenado, normalmente mediante una tabla hash, para guardar la info de un nodo procesado) para almacenar el mejor camino encontrado desde cada estado hasta el inicial, ampliando al expandir nuevos nodos.
Se pueden producir reorientaciones, si se encuentra un camino mejor para un nodo **N:**

- Si **N** esta en **ABIERTA**, se anota el nuevo camino mejor en **TABLA_A**
- Si **N** ya se ha expandido, se extiende el proceso recursivamente a sus hijos**.**

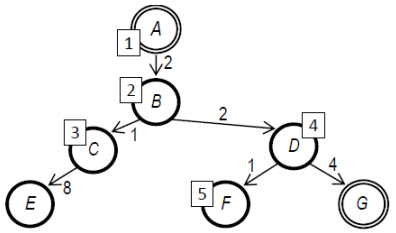
Recorrido de árboles

Búsqueda en amplitud: Se utiliza una **cola** donde se meten los nodos generados. Garantiza la solución de menor coste (óptima), si existe:

Búsqueda en profundidad: Busca expandir un único camino desde la raíz, si llega a un callejón sin salida, retrocede al nodo más cercano con alternativa. Se usa una **pila** para ir guardando los nodos generados. Suele existir un **límite de exploración** (máxima longitud que puede alcanzar cualquier camino desde la raíz).

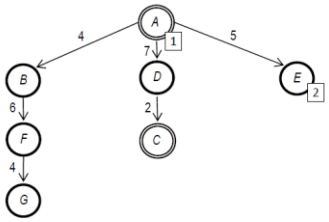
Métodos derivados de los anteriores: O también la **búsqueda bidireccional**, se busca desde el estado inicial a uno meta, y desde uno meta hasta un estado inicial. Ambas serán en amplitud, se garantiza que si existe un camino solución, en algún momento las dos búsquedas pasarán por un estado en común, desde donde se podrá construir ese camino.

Búsqueda de coste uniforme: Se va eligiendo en cada caso, el que tiene menor coste:

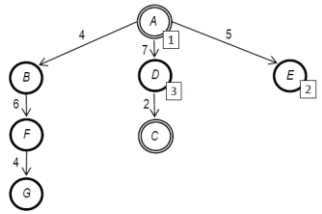


Búsqueda en anchura iterativa: Por ejemplo de derecha a izquierda:

Primera iteración

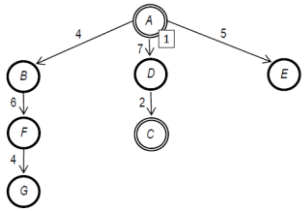


Segunda iteración

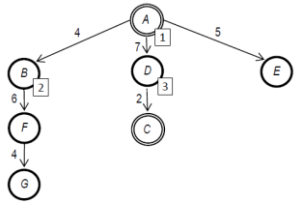


Búsqueda en profundidad iterativa: Por ejemplo de izquierda a derecha: Se expanden cómo máximo los nodos que no están a profundidad límite. Se puede encontrar el meta sin expandir

Primera iteración



Segunda iteración



Complejidades: En el caso peor. Hablamos de tamaño de **ABIERTA**

Algoritmo	Tiempo	Espacio
Anchura	$O(n^p)$	$O(n^p)$
Profundidad	$O(n^p)$	$O(n \cdot p)$
Coste uniforme (todas las reglas igual coste)	$O(n^p)$	$O(n^p)$
Profundidad iterativa	$O(n^p)$	$O(p)$
Anchura iterativa	$O(n^p)$	$O(n^p)$
Bidireccional (dos búsquedas en anchura)	$O(n^{p/2})$	$O(n^{p/2})$

n: "factor de ramificación" (número medio de sucesores de todos los nodos)

p: "profundidad de la solución"

Completud y admisibilidad:

Un algoritmo de búsqueda es **completo** si siempre que existe una solución la encuentra.

Un algoritmo es **admisible** (o **exacto**) si siempre encuentra una solución optima

En cuanto a propiedades de los algoritmos:

– **Primero en anchura**. Completo y admisible (si operadores del mismo nivel tienen = coste).

– **Primero en profundidad**. No completo (y, por tanto, no admisible).

– **Coste uniforme**. Admisible (y, por tanto, completo).

– **Búsqueda en profundidad iterativa**. Admisible.

– **Búsqueda en anchura iterativa**. Completo, pero no admisible.

– **Búsqueda bidireccional**. Completo si una de las búsquedas es en anchura.

Recorrido de grafos

Usamos el **Algoritmo general de búsqueda en grafos**.

Maneja dos listas **ABIERTA** y **CERRADA**.

En la primera, los nodos generados pero aún no expandidos. En la segunda, los nodos de **ABIERTA** seleccionados para su expansión.

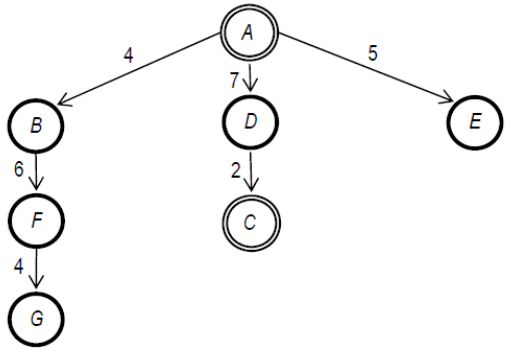
Si en cualquier momento se quiere reanudar un camino abandonado, se acude a la lista **ABIERTA** para poder hacerlo.

Cualquier algoritmo puede seleccionar nodos de **ABIERTA** con el criterio que quiera. Además, del grafo parcial que se va generando durante la ejecución, se gestiona un árbol cuyos enlaces indican caminos de menor coste encontrados en cualquier momento, de cada nodo del grafo parcial al raíz.

Se tienen en cuenta tres situaciones para expandir un nodo.

- Que el nodo generado no esté en **ABIERTA** ni en **CERRADA**. En ese caso, se enlaza el nodo generado con su padre
- Que el nodo esté en **ABIERTA**: Hay nuevo camino desde el nodo raíz hasta el nodo generado. Se ha de determinar si este nuevo camino es menos costoso que el anterior. Si es así, se redirige el puntero del árbol que parte del nodo generado.
- Que el nodo esté en **CERRADA**. Si el nuevo camino es menos costoso, además de llevar a cabo las operaciones del paso anterior, hay que hacer un estudio de los descendientes del nodo generado por si hubiera que redirigir enlaces actuales.

Contenido de abierta: Antes de expandir nodos



En Anchura por ejemplo

Antes de sacar A: {A}

Antes de sacar B: {B, D, E}

Antes de sacar D: {D, E, F}

Antes de sacar E: {E, F, C}

Antes de sacar F: {F, C}

Antes de sacar C: {C, G}

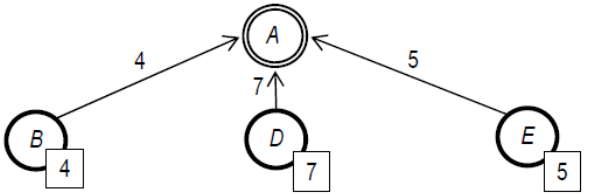
META ENCONTRADA

Contenido de TABLA_A: Después de expandir nodos.

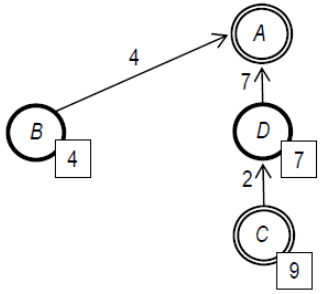
Ejercicios para describir el contenido de **TABLA_A** tras expandir cada nodo:

En profundidad por ejemplo:

- Inicialmente, *A* sería incluido en TABLA_A. Gráficamente esto se correspondería con un único nodo *A*. A continuación, expandimos el nodo inicial.



- Expandimos *E*. Seguidamente, aplicamos la función LimpiarTABLA_A a *E* por no tener hijos en ABIERTA y es sacado de TABLA_A. A continuación expandimos *D*:



- A continuación elegiríamos *C* para su expansión y llegaríamos a la meta. El camino hallado hasta la meta tiene coste 9.