P1 | Primer Parcial v2

Miguel Ángel Fernández Gutiérrez

A menos que se especifique lo contrario, las ilustraciones que aparecen han sido tomadas de

S. Russell, P. Norvig (2004). *"Inteligencia Artificial: un Enfoque Moderno"* (2nd ed.). Pearson-Prentice Hall.

Pregunta 1. El concepto de agente. Agentes racionales vs agentes inteligentes. Arquitecturas de agentes.

El concepto de agente

Un **agente** es un sistema (de ordenador) situado(1) en algún entorno, que es capaz de realizar acciones de forma autónoma(2) y que es flexible(3) para lograr los objetivos planteados.

(1) **Situación:** el agente recibe entradas sensoriales de un entorno en donde está situado y realiza acciones que cambian dicho entorno.

(2) **Autonomía:** el sistema es capaz de actuar sin la intervención directa de los humanos y tiene control sobre sus propias acciones y estado interno.

(3) **Flexibilidad:**

- **Capacidad estímulo-respuesta:** el agente debe percibir el entorno y responder de una forma temporal a los cambios que ocurren en dicho entorno.
- **Pro-activo:** los agentes no deben simplemente actuar en respuesta a su entorno, deben de ser capaces de exhibir comportamientos dirigidos a lograr objetivos, y tomar la iniciativa cuando sea apropiado.
- **Capacidad social:** los agentes deben de ser capaces de interactuar, cuando sea apropiado, con otros agentes artificiales o humanos para completar su propio proceso de resolución del problema y ayudar a otros con sus actividades.

La **secuencia de percepciones** de un agente refleja el historial completo del agente: todo lo que ha percibido. En un momento dado, el agente decidirá de acuerdo a este historial, ya sea completo o una parte de éste. En términos matemáticos, podemos decir que el comportamiento de un agente viene dado por la **función del agente**, que asocia una acción a cada percepción.

Agentes racionales vs agentes inteligentes

Primeramente, hablaremos brevemente del **test de Turing**. Esta prueba, propuesta por Alan Turing, se diseñó para proporcionar una definición de inteligencia más operacional: en lugar de proporcionar una lista de cualidades que consideramos "inteligentes", un ordenador pasará la prueba si un evaluador humano no es capaz de distinguir si las respuestas a una serie de preguntas planteadas son o no de un humano.

Un **agente inteligente** será un agente capaz de pasar el test de Turing, mientras que, por otra parte, un **agente racional** es aquel que en una situación intenta elegir, para cada una de las acciones posibles, aquella que tenga un mayor beneficio (informalmente, podríamos decir que la "mejor de las opciones posibles").

Es claramente más factible construir un agente racional.

Arquitecturas de agentes

De acuerdo a su **topología**, podemos distinguir entre:

- **Arquitecturas horizontales**, en las que todas las capas del agente son capaces de percibir estímulos y realizar acciones.
- **Arquitecturas verticales**, en las que una primera capa se encarga de percibir el entorno, y cada capa transmite información a la siguiente, tras llegar a una última capa que es la encargada de realizar acciones (podríamos decir que transmiten la información de forma "lineal").
- **Arquitecturas híbridas**, en la que la capa que percibe el entorno también realiza las acciones, y todas las capas transmiten y reciben información, en forma de cadena.

De acuerdo a su **nivel de abstracción**, podemos distinguir entre:

- **Arquitecturas reactivas:** actúan en función del entorno, que perciben por sus sensores. Operan sin necesidad de representar su entorno.
- **Arquitecturas deliberativas:** cuentan con un modelo del entorno, que usan para planificar acciones para llegar al objetivo.
- **Arquitecturas híbridas:** mezclan ambas arquitecturas anteriores.

Pregunta 2. Características de los agentes reactivos y deliberativos. Similitudes y diferencias. Arquitecturas.

Características de los agentes reactivos y deliberativos

Un **agente reactivo** es aquel que no tiene una representación interna sobre el mundo en el que se sitúa. A través de ciertos sensores percibe el mundo, analiza la información percibida y, conforme a ésta, realiza una acción determinada. Su comportamiento es diseñado de forma que pueda anticiparse a todas las posibles situaciones de su entorno.

Un **agente deliberativo** es aquel que trabaja por medio de la deducción lógica, a partir de un modelo conceptual de la realidad en la que se encuentra, que el agente almacena. Estos agentes analizan y elaboran un plan para lograr su objetivo.

Similitudes y diferencias

Como similitudes, observamos que ambos son agentes racionales. Sin embargo, y como hemos comentado, un agente reactivo no tiene ningún modelo del mundo, al contrario que un agente deliberativo. Por otra parte, un agente reactivo es más simple que un agente deliberativo.

Además, una de las diferencias principales estriba en el "razonamiento": mientras que un agente deliberativo razona sobre el entorno en el que actúa, percibiendo y planificando acciones (y pudiendo modificar sus acciones a medida que las va ejecutando), un agente reactivo se comporta en función de lo que encuentra en su camino – simplemente percibe y actúa.

Por otra parte, los agentes reactivos usan arquitecturas horizontales, mientras que los agentes deliberativos usan arquitecturas verticales.

Arquitecturas

Algunas arquitecturas para la construcción de agentes reactivos son:

- **Sistemas de producción.** Se comprueban secuencialmente reglas y se selecciona la primera que se verifique, ejecutando las acciones asociadas (que pueden ser una o varias acciones primitivas o la llamada a otro sistema de producción).

$$c_1 \to a_1 \quad c_2 \to a_2 \quad \cdots \quad c_i \to a_i \quad \cdots \quad c_m \to a_m$$

- **Redes neuronales.** Redes de unidades lógicas con umbrales.

- **Arquitecturas de subsunción.** Consisten en agrupar **módulos de comportamiento**. Cada módulo de comportamiento tiene una acción asociada, recibe la percepción directamente y comprueba una condición. Si esta se cumple, el módulo devuelve la acción a realizar. Un módulo se puede subsumir en otro. Si el módulo superior del esquema se cumple, se ejecuta este en lugar de los módulos inferiores.

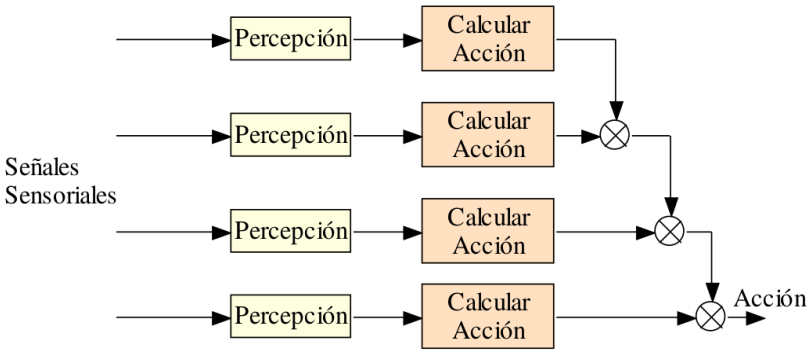


Ilustración tomada de las transparencias de la asignatura

- **Agentes reactivos con memoria.** Mejoran la precisión teniendo en cuenta el *historial* sensorial previo. La representación de un estado en el instante $t+1$ es función de la entradas sensoriales en el instante $t+1$, la representación del estado en el instante anterior t y la acción seleccionada en el instante anterior t.

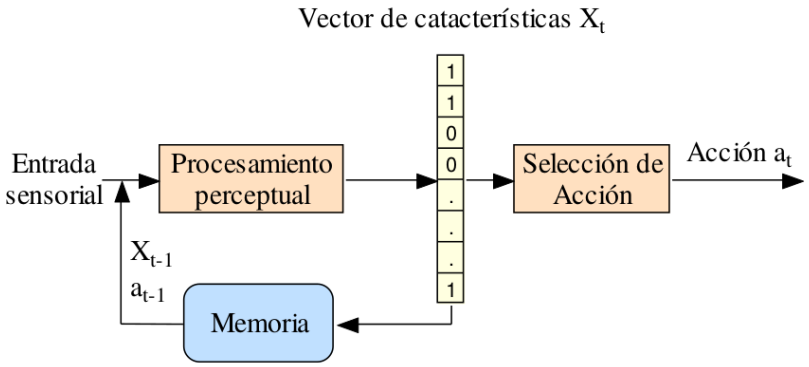


Ilustración tomada de las transparencias de la asignatura

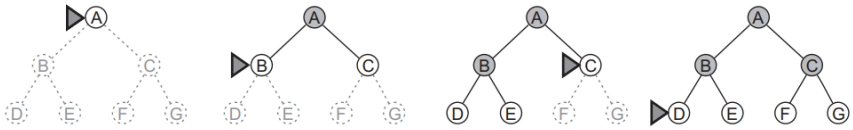
La memoria puede implementarse con:

- **Sistemas basados en pizarras.** Son extensiones de los sistemas de producción. En el agente existen varios programas denominados **módulos de conocimiento** (MC), formados por una parte de condición y otra parte de acción, y existe una memoria común a todos los MC denominada **pizarra**. Cada MC es "experto" en una parte concreta del problema a resolver, y cuando se cumple su condición, un MC puede actualizar la pizarra, realizar una acción concreta o ambas.

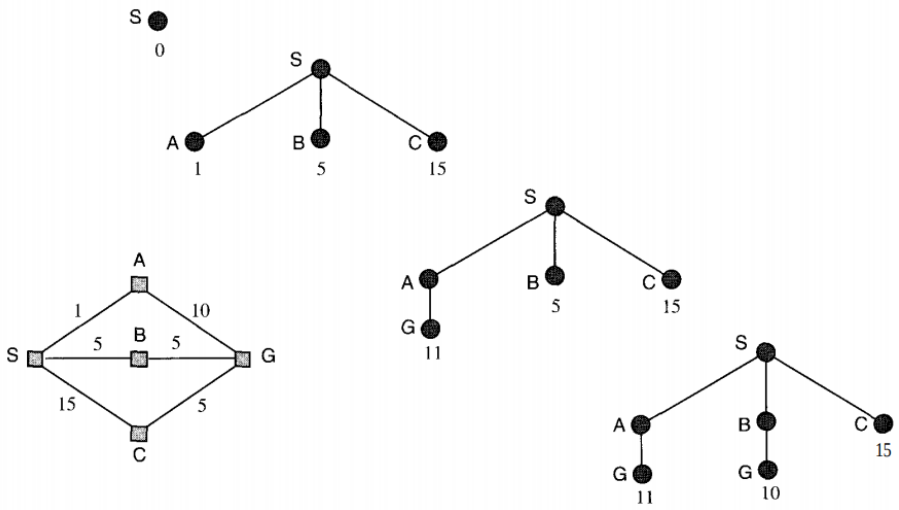
Pregunta 3. Describir brevemente los métodos de búsqueda no informada.

Describiremos brevemente a continuación estos métodos de búsqueda. Todos consisten en encontrar un camino desde un objetivo a un destino, recorriendo el **árbol de estados** del problema.

- **Búsqueda primero en anchura:** desde el estado inicial, analizamos todos los sucesores de cada nodo antes de pasar al siguiente nivel de búsqueda, hasta llegar a un nodo objetivo. De este modo, encontramos la ruta más "corta" (la que requiere pasar por menos nodos).

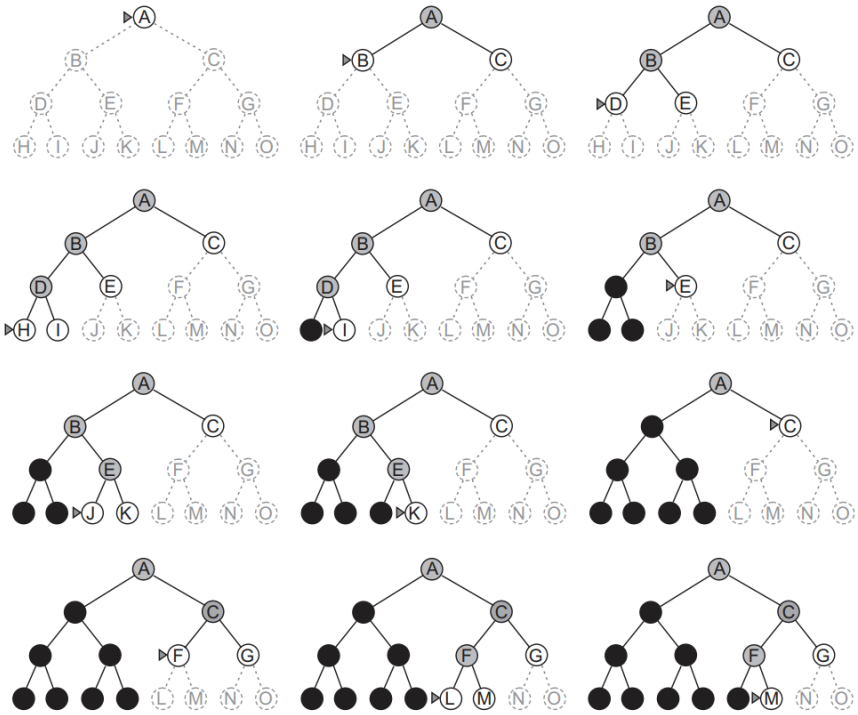


- **Búsqueda de costo uniforme:** expandimos el nodo con menor coste de todos los nodos de la frontera. Obtenemos de esta forma la solución con menor costo.

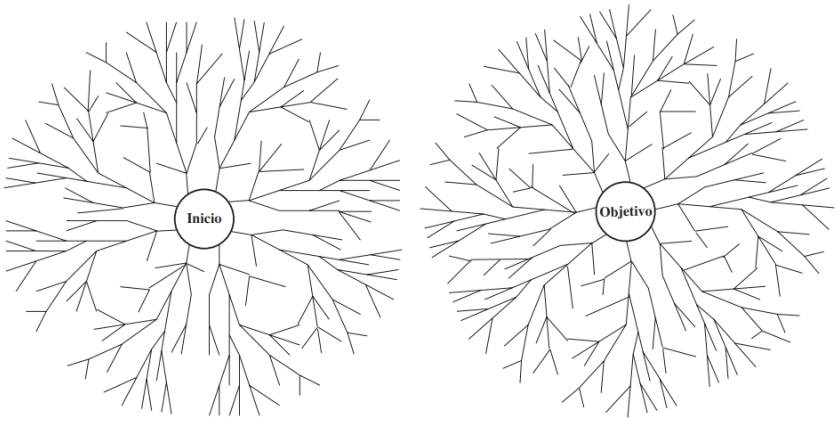


- **Búsqueda primero en profundidad:** desde el estado inicial, analizamos el sucesor del nodo de mayor nivel analizado hasta el momento. Esta búsqueda no nos garantiza ni la ruta más "corta" (en el sentido antes descrito) ni la de menor costo, simplemente una solución.

    - **Búsqueda en profundidad limitada:** evita la posibilidad de entrar "pozos sin fondo". Consideremos que la primera rama es infinita, o tiene un gran tamaño, y que la solución está en otra rama. Esta búsqueda tardará mucho en encontrar una solución, si es que la encuentra. Esta modificación del algoritmo de búsqueda en profundidad consiste en indicar un número máximo de niveles que puede descender.
    - **Búsqueda en profunidad iterativa:** es una modificación del algoritmo anterior, en el que la cota de profundidad se va aumentando si no se encuentra la solución.



- **Búsqueda bidireccional:** se ejecutan dos búsquedas simultáneas, una hacia delante desde el nodo inicial, y otra hacia atrás desde el objetivo, parando cuando ambas búsquedas se encuentren (el nodo expandido de un árbol esté en la frontera del otro).



> **Pregunta 4.** El concepto de heurística. Cómo se construyen las heurísticas. Uso de las heurísticas en IA.

Concepto de heurística

Las **heurísticas** son criterios, métodos o principios para decidir qué cauce, de entre una serie de cauces alternativos de acción, promete ser más efectivo a la hora de lograr una meta.

Esto representa un compromiso entre dos exigencias: la necesidad de que tales criterios sean simples y, al mismo tiempo, que puedan discriminar correctamente entre buenas y malas opciones.

Construcción de heurísticas

La construcción de funciones heurísticas puede considerarse un proceso de descubrimiento, pues es difícil articular el mecanismo por el que llegar a estas funciones. Como regla general, construimos las heurísticas a través de modelos simplificados del dominio del problema a resolver.

Uso de las heurísticas en IA

En Inteligencia Artificial, especialmente en la basada fuertemente en búsqueda, los métodos heurísticos siempre han sido utilizados. Una heurística encapsula el conocimiento específico que se tiene sobre un problema, y sirve de guía para que un algoritmo de búsqueda pueda encontrar una solución.

Los métodos heurísticos, a pesar de que no garantizan la solución óptima de forma general, producen en media resultados satisfactorios en la resolución de un problema. De hecho, una heurística puede devolver soluciones óptimas bajo ciertas condiciones (habrá que demostrarlo, evidentemente).

Algunas de las heurísticas más utilizadas son:

- La **distancia Manhattan**, para mapas en las que nos movemos en cuatro direcciones.
- La **distancia Manhattan adaptada**, para el caso de seis direcciones (mapas hexagonales).
- La **distancia diagonal**, para el caso de ocho direcciones.
- La **distancia Euclídea**, que inicialmente podría pensarse una buena heurística cuando podemos movernos en cualquier dirección, pero que no suele ser una buena opción.

Pregunta 5. Los métodos de escalada. Caracterización general. Variantes.

Métodos de escalada

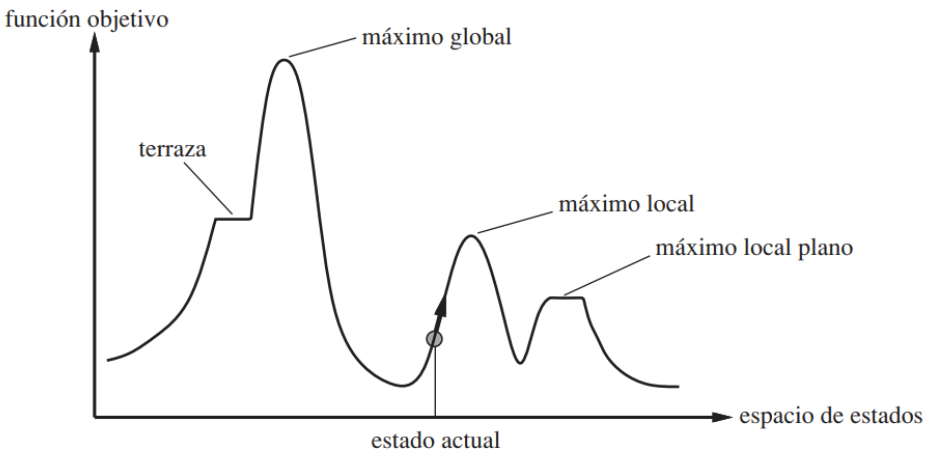
Los **métodos de escalada** son métodos que buscan en un entorno local del nodo en curso. Una búsqueda local consiste en seleccionar la mejor solución en el vecindario (entorno) de una solución inicial, e ir viajando por las soluciones hasta encontrar un óptimo (local o global). Estos métodos se diferencian en cómo determinan la vecindad de cada nodo.

Normalmente, estos métodos son usados cuando no es relevante el camino objetivo, sino llegar a éste. Como sólo almacenan la información de los nodos vecinos, suelen ser más eficientes en memoria.

Caracterización general

Las principales características de los métodos de escalada son:

- **Informados:** utilizan información del estado para elegir un cierto nodo.
- **No completos:** la existencia de máximos locales, crestas y mesetas (o terrazas) hacen que este algoritmo no siempre alcance la solución óptima.



- **Poco uso de memoria:** usan muy poca memoria, generalmente una cantidad constante.
- **No admisibles:** al no ser completos, tampoco serán admisibles.
- **Eficiencia:** mejora considerablemente si la función es monótona.
- La localidad evita que una parte del espacio de estados sea explorada.
- Pueden encontrar soluciones razonables en espacios de estados muy grandes.

Variantes

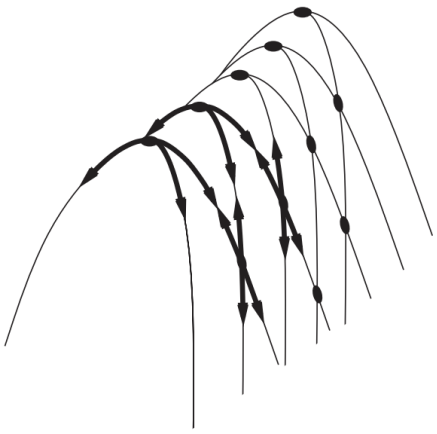
- **Método de escalada simple:** considera que la vecidad de un nodo es un conjunto de hijos obtenido secuencialmente hasta que aparece un nodo mejor que el padre ("mejor" según la función de evaluación). El algoritmo es:

    1. Se parte de un nodo inicial.
    2. Para cada nodo en curso: se obtiene la vecindad (según el criterio anterior).
        2.1. Si no se obtiene un hijo mejor que el padre: *parar*.
        2.2. En caso contrario: el hijo pasa a ser el nodo en curso, volver a 2.

- **Método de escalada por la máxima pendiente (*hill climbing*):** considera que la vecindad de un nodo es el conjunto de todos sus hijos obtenidos secuencialmente. Comienza con una solución arbitraria a un problema, e intenta encontrar una solución mejor variando incrementalmente un único elemento de la solución. Se van realizando cambios incrementales a las nuevas soluciones hasta que no pueden encontrarse mejoras. Expresado de otra forma, el algoritmo es:

    1. Se evalúa el estado inicial. Si también es el estado objetivo, devolverlo y terminar. En caso contrario, continuar con el estado inicial como estado actual.
    2. Partir de la solución actual. Expandirla.
        2.1. Buscar el hijo de mejor calidad según la función de evaluación.
        2.2. Si es "mejor" que la solución actual entonces sustituir la solución actual por dicho nodo. Volver a 2.
        2.3. Si no se encuentra un hijo mejor que el padre: *parar*.

- **Método de escalada estocástica:** escoge aleatoriamente entre los sucesores con mejor valoración que el estado actual. Este método resuelve el **problema de ascensión de colinas**, como vemos en la ilustración siguiente, en la que la rejilla de estados (círculos oscuros) se pone sobre una cresta que se eleva de izquierda a derecha y crea una secuencia de máximos locales que no están directamente relacionados el uno con el otro (de cada máximo local, todas las acciones posibles se señalan hacia abajo).



Como este método escoge aleatoriamente de entre los movimientos ascendentes (los que tienen mejor valoración que el estado actual), y haciendo que la probabilidad de selección pueda variar con la pendiente del movimiento ascendente, da generalmente mejores soluciones.

- **Método de escalada de primera opción:** Se generan aleatoriamente sucesores, escogiendo el primero con mejor valoración que el estado actual.
- **Método de escalada de reinicio aleatorio:** Se repite varias veces la búsqueda, partiendo cada vez de un estado inicial distinto, generado aleatoriamente. En otras palabras, "si no te sale a la primera, inténtalo otra vez". Si la probabilidad de éxito en la búsqueda es p, entonces el número de reinicios esperado es $1/p$.
- **Algoritmo de enfriamiento simulado:** es un modo de evitar que la búsqueda local finalice en máximos locales, permitiendo que algunos movimientos sean hacia soluciones peores. Si la búsqueda avanza realmente a una buena solución, estos movimientos de *escape de óptimos locales* deben realizarse de forma controlada.

  En resumidas cuentas, este algoritmo parte de una temperatura inicial, y en cada iteración escoge un movimiento aleatorio. Si este movimiento mejora la situación, es aceptado; en caso contrario, la probabilidad de que sea escogido dependerá de la *maldad* del movimiento (cuanto más malo sea, menor probabilidad) y de la temperatura en cada momento (a mayor temperatura, mayor probabilidad).

Pregunta 6. Características esenciales de los algoritmos *primero el mejor*.

Los **algoritmos de búsqueda *primero el mejor*** son aquellos que avanzan a través del mejor nodo encontrado hasta el momento (escogido de acuerdo a la función de evaluación). Para ello usamos:

- **Lista de nodos abiertos.** Son nodos que se han genenerado y los que se les ha aplicado la función heurística, pero cuyos sucesores aún no han sido generados.
- **Lista de nodos cerrados.** Son nodos que ya han sido examinados. Se usan para ver si un nodo ya ha sido generado con anterioridad.

La **función de evaluación** devuelve una cantidad que sirve para representar lo deseable o indeseable que será la expansión de cada nodo.

Un algoritmo de búsqueda primero el mejor es como sigue:

Algoritmo 6.1 (Primero el mejor).

$G \leftarrow$ crear grafo de búsqueda
$I \leftarrow$ nodo inicial
inicializamos las listas:
    abiertos $\leftarrow \{I\}$; *cerrados* $\leftarrow \{\}$; *éxito* $\leftarrow$ falso
hasta que (*abiertos* está vacío) o (*éxito*)**:**
    $N \leftarrow$ primer nodo de *abiertos*
    quitar N de *abiertos*
    añadir N a *cerrados*
    si N es estado final**:** *éxito* $\leftarrow$ verdadero
    en caso contrario:
        $S \leftarrow$ conjunto de sucesores de N que no son antecesores de N en G
        por cada s **en** S**:**
            generar un nodo en G
            si s no está ya en G**:**
                establecer un puntero a N desde s
                añadir s a *abiertos*
            si s está en *abiertos* o en *cerrados***:**
                decidir si redirigir sus punteros hacia N
            si s está en *cerrados***:**
                decidir si redirigir los punteros de los nodos en sus subárboles
        reordenar *abiertos* según la función de evaluación
si *éxito***:** *solución* $\leftarrow$ camino desde I hasta N a través de los punteros de G
en caso contrario: *solución* $\leftarrow$ fracaso

Pregunta 7. Elementos esenciales del algoritmo A*.

El **algoritmo A*** es un algoritmo que se nutre del valor heurístico de los nodos y el coste real del recorrido, usando por tanto la siguiente función de evaluación para cada nodo n:

$$f(n) = g(n) + h(n)$$

- $g(n)$ es el **coste real** del camino recorrido hasta llegar al nodo n desde el nodo inicial.
- $h(n)$ es el **valor heurístico** (estimado) desde n hasta el objetivo.

El algoritmo mantiene dos estructuras de datos, una **lista de abiertos** y una **lista de cerrados**, que ya comentamos en la pregunta anterior. Para que el nodo a expandir sea el mejor, implementamos la lista de abiertos con una cola con prioridad. En cada paso del algoritmo, expandimos el primer nodo de *abiertos*, y en caso de que no sea el objetivo, calculamos la $f(n)$ para cada uno de sus hijos, y los vamos insertando en *abiertos*, y pasamos este nodo a *cerrados*. Si el mejor de los nodos abiertos es el objetivo, acaba, y si no continúa. Detallaremos el algoritmo a continuación.

Algoritmo 7.1 (A*).

$I \leftarrow$ nodo inicial
$G \leftarrow$ crear grafo de búsqueda, inicializado con I
inicializamos las listas:
    abiertos $\leftarrow \{I\}$; *cerrados* $\leftarrow \{\}$; *éxito* $\leftarrow$ falso
hasta que (*abiertos* está vacío) o (*éxito*)**:**
    $N \leftarrow$ primer nodo de *abiertos*
    quitar N de *abiertos*
    añadir N a *cerrados*
    si N es estado final**:** *éxito* $\leftarrow$ verdadero
    en caso contrario:
        $M \leftarrow$ conjunto de sucesores de N que no son ancestros de N en G
        añadir M a los sucesores de N en G
        por cada m **en** M**:**
            generar un nodo en G
            si m no está ya en *abiertos* ni en *cerrados***:**
                establecer un puntero a N desde m
                añadir m a *abiertos*
            si m está en *abiertos* o en *cerrados***:**
                modificar el puntero para que apunte a N siempre que el mejor
                    camino encontrado hasta el momento hacia dicho nodo
                    pasa por N

> **si** m está en *cerrados*:
> modificar los punteros de cada uno de sus descendientes de modo que apunten hacia atrás a los mejores caminos encontrados hasta el momento a esos descendientes
> reordenar *abiertos* según la función de evaluación, $f(n)$, de menor a mayor (en caso de empate, emplear el criterio de profundidad en el árbol de búsqueda)
> **si** *éxito*: *solución* $\leftarrow$ camino desde I hasta N a través de los punteros de G
> **en caso contrario:** *solución* $\leftarrow$ fracaso

Este algoritmo es **completo**, es decir, siempre dará con la solución en caso de que exista. Además, hemos de notar que los algoritmos de búsqueda voraz y de Dijkstra son casos particulares del algoritmo A*, cuando $g(n) = 0$ y $h(n) = 0$ para cada nodo, respectivamente.

Para que el algoritmo alcance la solución óptima, la heurística h deberá ser **admisible**, es decir, una heurística que no sobreesitme el coste real de alcanzar el nodo objetivo. Dicho de otra forma, si llamamos $\hat{h}(n)$ al coste del camino óptimo desde el nodo n hasta la solución óptima, una heurística admisible deberá verificar, para cada nodo n:

$$h(n) \leq \hat{h}(n)$$

Además, diremos que la heurística h_2 está **mejor informada** que la heurística h_1 si su estimación es más cercana a la óptima, esto es, si se cumple para cada nodo n que

$$0 \leq h_1(n) \leq h_2(n) \leq \hat{h}(n)$$

Un algoritmo A* será mejor en cuanto más informada esté su heurística.

Finalmente, diremos que una heurística h es **consistente** si, dados dos nodos n_1 y n_2 tales que n_2 es sucesor de n_1, se verifica

$$h(n_1) - h(n_2) \leq c(n_1, n_2)$$

donde $c(n_1, n_2)$ es el coste del arco que une n_1 con n_2. Si se cumple la condición de consistencia para h, entonces f es no decreciente, es decir, $f(n_1) \leq f(n_2)$ para cualquier par de nodos n_1, n_2, con n_2 sucesor de n_1. Esto implica que, cada vez que A* expande un nodo, ha encontrado un **camino óptimo** al mismo.

Pregunta 8. Elementos esenciales de un algoritmo genético.

Los **algoritmos genéticos** son métodos sistemáticos para la resolución de problemas de búsqueda y optimización que aplican a éstos los métodos de la evolución biológica. En un proceso de evolución, existe una **población** de individuos. Los más adecuados de su entorno sobreviven, se **reproducen** y tienen descendencia (a veces con **mutaciones** que mejoran su idoneidad al entorno), mientras que los menos aptos son **descartados**.

Desde un punto de vista general, el objetivo de cualquier algoritmo genético es encontrar una solución óptima para una cierta función objetivo. Para ello, contamos con los siguientes elementos:

- **Cromosoma:** vector representación de una solución al problema.
- **Fenotipo:** conjunto de soluciones a un problema (cromosomas).
- **Gen:** característica, variable o atributo concreto del vector de representación de una solución.
- **Genotipo:** representación de los cromosomas mediante cadenas de dígitos binarios.
- **Generaciones:** iteraciones.
- **Población:** conjunto de soluciones al problema.
- **Adecuación al entorno o *fitness*:** valor de la función objetivo.
- **Selección natural:** operador de selección.
- **Mutación:** operador de modificación.
- **Cambio generacional:** operador de reemplazamiento.

Los algoritmos genéticos operan siguiendo el siguiente esquema general:

1. Generamos una *población* inicial de forma aleatoria (conformada por *cromosomas*).
2. Aplicamos a cada *cromosoma* una función de *adecuación al entorno*.
3. Repetimos el siguiente bucle, ya sea mediante un determinado número de iteraciones o hasta llegar a un cierto valor umbral (cuando un *cromosoma* supere este umbral pararemos el algoritmo):
    3.1. Fase de *selección*: seleccionamos los individuos que se cruzarán. Cuanto mayor sea su adecuación, mayor probabilidad tendrán de ser seleccionados.
    3.2. Fase de *cruce*: generamos un descendiente que tiene características combinadas de sus progenitores.
    3.3. Fase de *mutación*: cambiamos partes que no se hayan visto alteradas por el cruce.
    3.4. Fase de *reemplazo*: los mejores cromosomas pasarán a ser la nueva población.