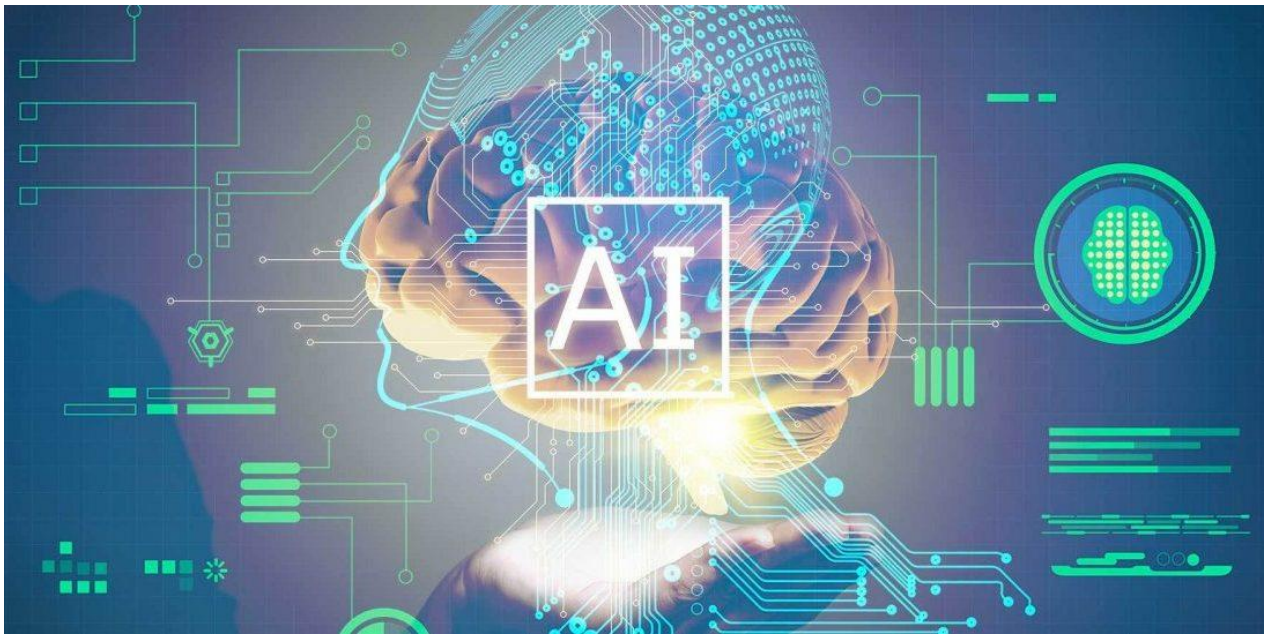


INTELIGENCIA ARTIFICIAL

TEMARIO UNIVERSIDAD UAH

DAVID MARQUEZ MIGNUEZ

HISTORIA DE LA INTELIOGENCIA ARTIFICIAL

La Inteligencia Artificial no ha nacido de la nada, sino que hay multitud de disciplinas que han construido ideas, punto de vista y técnicas para la IA. Entre todas ellas vamos a ver algunas y las cuestiones que plantean cada una de ellas.

FILOSOFIA

Presenta las siguientes cuestiones:

1. ¿Pueden las reglas formales ser usadas para llegar a conclusiones validas?
2. ¿Cómo surge la mente de un cerebro físico?
3. ¿De dónde viene el conocimiento?
4. ¿Cómo el conocimiento lleva a la acción?

Aristóteles fue el primero en formular silogismos para el razonamiento apropiado, que en principio permitía generar conclusiones dadas unas premisas iniciales. Mas tarde Raimundo Lulio tuvo la idea de que el razonamiento podría ser llevado a cabo por un artefacto mecánico. Se hicieron multitud de máquinas calculadoras que hicieron pensar que tenían un comportamiento más cercano al humano que al animal. Una cosa es decir que la mente opera (al menos en parte) siguiendo reglas lógicas y construir maquinas que puedan emular esa mente, pero otra cosa es decir que la mente en si es un sistema físico. Descartes distinguió entre mente y cuerpo y los problemas de esa distinción. Descartes era filosofo del **racionalismo**, pero también propuso el **dualismo** (parte de la mente humana esta fuera de la naturaleza). Una alternativa al dualismo era el **materialismo** (que dice que la mente es constituida por las operaciones del cerebro de acuerdo a leyes físicas). El movimiento del **empirismo** propuso el principio de **inducción**: donde hay unas reglas generales que se usan para ver asociaciones entre sus elementos. Mas tarde se desarrolló el **positivismo lógico** que decía que todo conocimiento era caracterizado por teorías lógicas conectadas. Mas tarde, la **teoría de la confirmación** intento analizar la adquisición de conocimiento a través de la experiencia. De los últimos aspectos filosóficos que influyeron en la IA vamos a ver es la **conexión entre conocimiento y acción** (cuestión muy importante para la IA), solo conociendo como las acciones son justificadas podemos entender cómo construir un agente cuyas acciones sean justificables. Aristóteles dijo que las acciones son justificadas por una conexión lógica entre las metas y el conocimiento adquirido en el resultado de dichas acciones.

MATEMATICAS

Presenta las siguientes preguntas:

1. ¿Cuáles son las reglas formales para concluir cosas validas? -> Lógica
2. ¿Qué puede ser computable? -> Computación
3. ¿Cómo actuamos ante información incierta? -> Probabilidad

La filosofía introdujo algunos fundamentos de la IA, pero un salto a una ciencia formal requiere un nivel de matemáticas en 3 áreas: lógica, computación y probabilidad.

La lógica tiene una larga historia (antigua Grecia) pero su desarrollo matemático no comenzó hasta el siglo XIX por George Boole que introdujo la lógica booleana. Mas tarde otros

matemáticos extendieron esa lógica llegando a la lógica que conocemos hoy día. El siguiente paso era ver que se podía hacer con la lógica y la computación: aparece el concepto **algoritmo** que se usa para realizar deducciones lógicas. Gödel mostro que existe procedimientos efectivos para probar deducciones verdaderas usando una lógica sencilla, pero esa lógica sencilla no podía capturar el principio matemático de inducción necesario para caracterizar los números naturales. Gödel demostró que hay límites en la deducción, su **teorema de la incompletitud** demostró que cualquier teoría formal por fuerte que fuese, existen deducciones que no pueden probarse. Esto nos indica que existen algunas funciones de los números enteros que no pueden ser representadas por algoritmos, es decir, que no podían ser **computables**. Turing con su máquina intentó mostrar que funciones eran computables (una noción algo confusa ya que no hay una definición formal).

Otro concepto importante que introdujo las matemáticas era la **trazabilidad**, es decir, si un problema puede o no resolverse en un espacio de tiempo medianamente razonable. ¿Cómo podemos reconocer problemas intrazables? La teoría de **NP-completitud** proporciona un método para ello.

El último de los conceptos introducidos a la IA por las matemáticas es el de la **probabilidad**. La probabilidad fue ideada inicialmente por Cardano y desarrollada por otros como Pascal, Fermat, Bernoulli, Laplace, etc.…

NEUROLOGIA

Presenta la siguiente pregunta:

¿Cómo el cerebro procesa información?

Desde tiempos de Aristóteles se sabía que el cerebro humano era diferente. No es hasta el siglo XIX cuando Paul Broca demostró la existencia de áreas especializadas de ciertas acciones en el cerebro, tales como hablar. Santiago Ramon y Cajal junto Rashervsky fueron los primeros en aplicar modelos matemáticos en el estudio del sistema nervioso. Estos y otros muchos estudios

ECONOMIA

Las preguntas que presenta la economía son las siguientes:

1. ¿Qué decisiones tomamos para maximizar el resultado?
2. ¿Cómo hacemos esto cuando otros no están de acuerdo?
3. ¿Cómo hacemos esto cuando el resultado es lejano en el tiempo?

Cuando McDonnals ofrece una hamburguesa a un dólar, ellos entienden que prefieren ganar un dólar y que los clientes quieren una hamburguesa. El tratamiento matemático de los **resultados preferidos o utilidades** fue normalizado por gente como Walras y desarrollado por Ramsey, Von Neumann, etc.…

La **teoría de la decisión**, que combina la teoría de la probabilidad y la teoría de la utilidad, ofrece un marco formal de decisiones para ser tomadas cuando hay incertidumbre. Esto es más útil para grandes economías cuando los agentes de esta no se fijan en las acciones individuales de otros agentes. Para economías pequeñas la situación es más como un **juego** donde las acciones de un jugador puede afectar significativamente a otro jugador.

Von Neumann y Morgenstern desarrollaron la **teoría de juegos**. A diferencia de la teoría de decisiones, la teoría de juegos no ofrece una solución inequívoca para seleccionar acciones.

PSICOLOGIA

Responde a la siguiente pregunta:

¿Cómo los humanos y animales piensan y actúan?

A lo largo de la historia ha habido multitud de investigadores y corrientes que han estudiado el comportamiento y pensamiento de los animales, pero sin llegar a ahondar mucho en el humano. La **psicología cognitiva**, que ve el cerebro como un dispositivo que procesa información, puede remontarse hasta los estudios de William James. Estos estudios llevaron a la creación de la **ciencia cognitiva**.

INTELIGENCIA COMPUTACIONAL

Responde a la siguiente pregunta:

¿Cómo podemos crear un computador eficiente?

Para el desarrollo de la IA se necesitan dos componentes: inteligencia y un artefacto donde aplicarla. Este artefacto es el computador. El computador se remonta a la II guerra mundial y su evolución ha sido clave para la IA.

EJERCICIOS

1.- Pregunta de examen: Campos de aplicación de la Inteligencia Artificial. Éxitos y fracasos de la misma.

2.- Pregunta de examen: Describa porque se necesito precisar el concepto de computabilidad y como repercutió en los intentos de desarrollar sistemas inteligentes.

3.- Pregunta de examen: Describa brevemente las disciplinas que han intervenido en el desarrollo de la IA y la computación así como las relaciones entre ambas.

Algunas de las disciplinas que han incidido en la IA son las siguientes. En primer lugar la filosofía la cual trata de resolver cuestiones acerca de la mente, creando una relación entre conocimiento y acción. Otra disciplina relacionada con la IA es la economía la cual trata de resolver cuestiones acerca de la preferencia o de la toma de decisiones de un individuo, desarrollando así una teoría de juegos. También está la matemática la cual trata cuestiones acerca de la lógica, la computabilidad y la probabilidad. Otra disciplina es la psicología que trata de resolver cuestiones acerca de del pensamiento tanto humano como animal, a partir de ahí se crea la ciencia cognitiva. Así se crean varias disciplinas que crean teorías alrededor de la IA.

4.- Pregunta de examen: Disciplinas que tiene incidencia en la IA.

Existen varias disciplinas que han construido ideas, disciplinas y puntos de vista alrededor de la Inteligencia Artificial. Algunas de estas disciplinas son la filosofía, las matemáticas, la neurología,

economía… Dichas disciplinas se encargan de resolver preguntas referentes a la IA, ¿Cómo el conocimiento lleva a la acción?, ¿Qué decisiones tomamos para maximizar el resultado?...

5.-Pregunta de examen: Características de los problemas tratados en la IA.

Todas las cuestiones tratadas por las disciplinas alrededor de la Inteligencia Artificial tienen que ver con la toma de decisiones, en general, la relación que existen entre conocimiento y accion.

6.- Pregunta de examen: De algunos ejemplos de aplicaciones existentes de la IA.

Algunos ejemplos de aplicaciones de la IA son por ejemplo en relación a las búsquedas tanto informadas como no, además de la resolución de problemas de la manera mas eficiente posible. Por otro lado también podemos aplicar la Inteligencia Artificial a juegos alternos de dos jugadores donde se debe determinar un ganador. En general el uso de la Inteligencia Artificial se suele reducir a la toma de decisiones.

7.- Pregunta de examen: Que indica la hipotesis del símbolo físico y que relación tiene con los paradigmas simbólico o conexionista de la IA.

BUSQUEDA DESINFORMADA

En este temas vamos a tratar estrategias para resolver problemas de búsqueda sin información, o búsqueda a ciegas. Para resolver estos problemas, emplearemos lo que se denomina como búsqueda en espacio de estados. La búsqueda en espacio de estados consiste en transformar un problema en un grafo, es decir, convertir nuestro problema principal en uno que se pueda modelar. Para modelizar un problema debemos definir componentes:

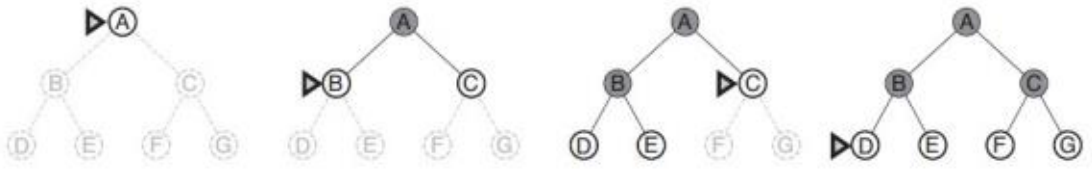
1. **Estado inicial:** será el punto donde comenzará nuestro problema.
2. **Operaciones de cambio de estado:** dado un estado, disponemos de operaciones para poder transitar a otros estados, por ejemplo, la acción operación(S,A) retorna el estado resultante de aplicar la operación A al estado S.
3. **Estado meta:** identificamos un estado en el que consideramos que hemos llegado a la solución.
4. **Coste:** una función que asigna un coste a cada camino según las operaciones que se hayan aplicado.

Dadas estas componentes, consideraremos como solución al problema una secuencia de operaciones que nos lleven desde un estado inicial hasta un estado meta. La calidad de la solución dependerá de la función de coste, es decir, del número de operaciones desde el estado inicial hasta el estado meta.

BUSQUEDA DE SOLUCIONES

Una vez disponeos de un problema, debemos resolverlo. Para ello, en primer lugar debemos realizar un ejercicio de abstracción, es decir, eliminar elementos del problema que no sean importantes. Una vez hemos abstraído el problema, tratamos de buscar una secuencia de acciones que desde un estado inicial nos permitan llegar hasta un estado final deseado. Todas estas secuencias de acciones conforman un árbol, de búsqueda siendo el estado inicial la raíz, las acciones son los vértices y los nodos corresponden a los estados. Suceden una secuencia de pasos:

1. Verificamos si estamos o no en un nodo solución.
2. Expandimos el estado actual, es decir, aplicamos todas aquellas operaciones disponibles desde dicho estado, generando así nuevos estados.
3. Elegimos cuál de los nuevos estados será el primero en contemplar, y volvemos a comenzar de nuevo.

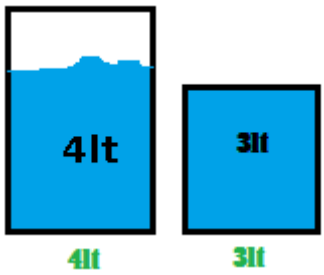


Esta estrategia presenta un problema, y son los bucles. Pare evitar esto, debemos saber en todo momento en que nodo estamos almacenando así todos los nodos que ya se han visitado y han sido examinados.

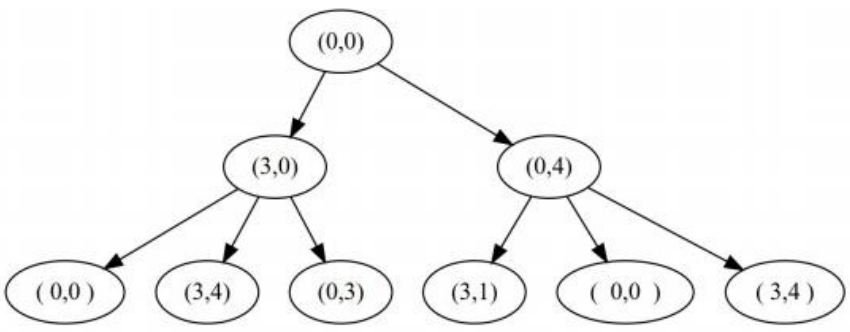
METODOS PARA EVALUAR UN ALGORTIMO DE BUSQUEDA

- **Completitud:** un algoritmo será completo si este nos garantiza que encontrará una solución si es que existe.
- **Optimo:** un algoritmo será optimo si este nos asegura que va a encontrar la solución más optima, es decir, aquella solución que nos proporcione una función de coste lo más baja posible.
- **Complejidad temporal:** es la cantidad de tiempo que el algoritmo tarda en encontrar la solución, depende del factor de ramificación y la profundidad de las soluciones.
- **Complejidad espacial:** es la cantidad de memoria que se necesita para la búsqueda, al igual que la complejidad temporal depende del factor de ramificación y de la profundidad de las soluciones.

Antes de comenzar a explicar los algoritmos de búsqueda, vamos a ver un ejemplo sencillo sobre como abstraer un problema de búsqueda desinformada y transformarlo en un problema de búsqueda en espacio de estados. En el problema disponemos de3 dos jarras, la primera jarra tiene una capacidad de 2L y la segunda una capacidad de 4L. El objetivo es que una de las jarras tenga 2L de líquido pudiendo solo vaciar o llenar las mismas.



En primer lugar vamos a abstraer el problema eliminando los elementos del mismo que no son relevantes para hallar la solución. En segundo lugar crearemos dos variables que representaran la cantidad de liquida que hay en cada jarra en un momento determinado. ¿Cuál sería el espacio de estados de este problema? Por ejemplo, en la jarra x de 3 litros sabemos que puede haber o bien 0 litros (es decir, está completamente vacía) o bien a su máxima capacidad (3 litros), pero también pueden quedar tanto 1 como 2 litros en la jarra intercambiando la bebida con la otra jarra de 4 litros. El estado al que queremos llegar será aquel el cual alguna de las dos jarras contenga 2L de líquido, este será nuestro estado meta. Podemos tener tanto uno o varios estados iniciales como uno o varios estados finales. En conclusión, la forma de resolver este tipo de problemas es mediante la construcción de un grafo, donde los estados representarán los nodos y las operaciones las aristas de dicho grafo.



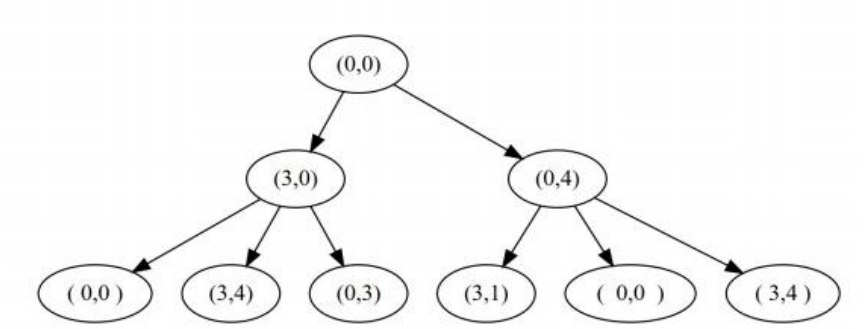
ESTRATEGIAS PARA LA BUSQUEDA DESINFORMADA:

BUSQUEDA EN ANCHURA

En la búsqueda en anchura, el nodo raíz se expande primero para luego expandirse todos sus sucesores. Normalmente, todos los nodos de un nivel del árbol son expandidos antes de que los nodos del siguiente nivel sean expandidos. En esta búsqueda, el nodo menos profundo es elegido para su expansión, para ello se gestiona la lista de ABIERTOS como una lista en orden FIFO. Por lo tanto, la primera vez que aparezca la meta será a través del camino más corto (menos profundo), sabemos que es la meta menos profunda ya que todos aquellos nodos que son menos profundos han sido explorados y han fallado el test de éxito.

La búsqueda en profundidad es un algoritmo completo pues si el nodo meta existe, dicho algoritmo la encontrara tarde o temprano. Además el algoritmo también es óptimo siempre y cuando los costes de los caminos para ir de nodo a nodo son todos iguales. En cuanto a la complejidad temporal, supongamos un árbol donde la solución se encuentra a profundidad p, y cada nodo tiene r sucesores, la complejidad temporal será del orden de $O(r^{p+1})$. Para calcular la complejidad espacial, debemos tener en cuenta que en la búsqueda en anchura, todo nodo generado permanece en memoria, la complejidad espacial será $O(r^{p})$.

Volviendo al problema de las jarras, podemos resolverlo de manera muy sencilla. Inicialmente ambas jarras estarán vacías:



ACTUAL	SUCESORES	ABIERTOS
		(0,0)

A continuación aplicamos las operaciones disponibles para el estado en el que nos encontramos. En este caso las únicas operaciones disponibles son llenar la jarra de 3L o llenar la de 4L.

ACTUAL	SUCESORES	ABIERTOS
		(0,0)
(0,0)	(3,0) (0,4)	(3,0) (0,4)

Tras expandir el nodo inicial, seleccionamos uno de los nodos por ejemplo (3,0) y procedemos como en el paso anterior. En este caso podemos vaciar la jarra de 3L, llenar la jarra de 4L y verter el contenido de la jarra de 3L en la jarra de 4L.

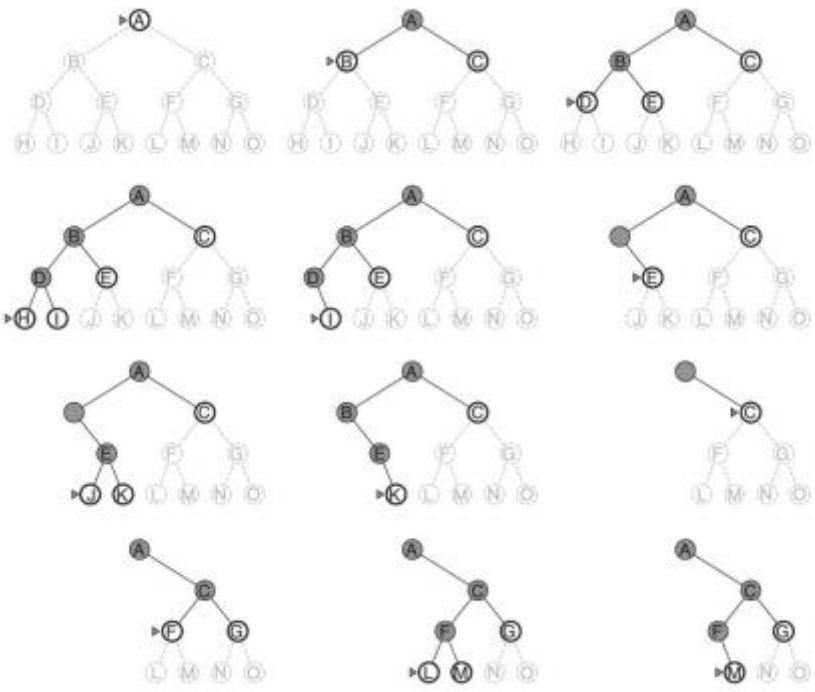
ACTUAL	SUCESORES	ABIERTOS
		(0,0)
(0,0)	(3,0) (0,4)	(3,0) (0,4)
(3,0)	(0,0) (3,4) (0,3)	(0,4) (0,0) (3,4) (0,3)

Sin embargo, puede ocurrir que un mismo nodo se repita dos o más veces. Y bien, la pregunta es, ¿ello implica la posible aparición de bucles en estos grafos? En realidad no sería un inconveniente si un mismo nodo aparece repetido más de una vez como final de un camino, puesto que cada vez que aparezca de nuevo un nodo se alcanzará a través de un camino más largo. Por tanto, no debemos preocuparnos si un mismo nodo se repite durante la descripción del camino del grafo. Además, se podría también evitar la aparición de repeticiones, añadiendo una columna de cerrados, si está ahí un nodo, al intentar añadir a la columna de abiertos no se añadiría, pero esto supone aumentar el coste, ya que trabajaríamos con dos listas a la vez.

ESTRATEGIAS PARA LA BUSQUEDA DESINFORMADA:

BUSQUEDA EN PROFUNDIDAD

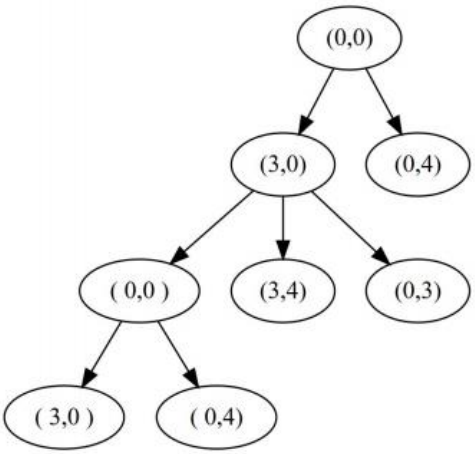
A diferencia de la búsqueda en anchura, en la búsqueda en profundidad siempre se expande el nodo más profundo en la lista de abiertos. El algoritmo llega hasta las hojas del árbol donde los nodos ya no tienen sucesores. En la búsqueda en profundidad, la gestión de la lista ABIERTOS se realiza como una pila o lo que es lo mismo, una lista con una política LIFO ultimo en entrar último en salir.



Por otra parte la búsqueda en profundidad no es un algoritmo completo, puesto que no puede asegurar obtener la solución debido a que pueden existir bucles que hagan que algunas ramas del árbol sean inaccesibles. Tampoco es óptimo puesto que se expande primero la rama de la izquierda independientemente que exista un nodo meta a la derecha.

En cuanto a la complejidad temporal, es mucho mayor que en la búsqueda en anchura y es del orden de $O(b^m)$ siendo m la profundidad máxima. Por otro lado la complejidad espacial es la parte buena que tiene este algoritmo La búsqueda en profundidad requiere de solo $O(bm)$ nodos en memoria, donde b indica el factor de ramificación y m la profundidad máxima.

Para entender mejor el algoritmo vamos a volver al problema de las jarras:



Al igual que antes inicialmente comenzamos con las jarras vacías (0,0) siendo este el nodo inicial.

ACTUAL	SUCESORES	ABIERTOS
		(0,0)

A continuación, pensamos las posibilidades para seguir expandiendo nuestro árbol, o bien llenamos hasta su capacidad máxima la jarra de 3L o bien la jarra de 4L.

ACTUAL	SUCESORES	ABIERTOS
		(0,0)
(0,0)	(3,0) (0,4)	(3,0) (0,4)

Tras expandir el nodo inicial, vayamos con el primero de los nuevos nodos (3,0), analizando los siguientes nodos, es decir, (0,0) (3,4) (0,3):

ACTUAL	SUCESORES	ABIERTOS
		(0,0)
(0,0)	(3,0) (0,4)	(3,0) (0,4)
(3,0)	(0,0) (3,4) (0,3)	(0,0) (3,4) (0,3) (0,4)

A continuación analizamos el primer nodo de la pila, en este caso el nodo (0,0):

ACTUAL	SUCESORES	ABIERTOS
		(0,0)
(0,0)	(3,0) (0,4)	(3,0) (0,4)
(3,0)	(0,0) (3,4) (0,3)	(0,0) (3,4) (0,3) (0,4)
(0,0)	(3,0) (0,4)	(3,0) (0,4) (3,4) (0,3) (0,4)

Como podemos observar a diferencia del algoritmo de búsqueda en anchura, la lista de abiertos se comporta como una pila, el primero que entra es el primero que sale.

Sin embargo, si nos fijamos en el primer nodo de la lista de abiertos (3,0), a la hora de encontrar sus nodos sucesores, nos damos cuenta de que una de las posibles operaciones es vaciar la jarra de 3 litros, quedándonos nuevamente en (0,0). A continuación, si examinamos los nodos sucesores de (0,0) nos percatamos de que una de las operaciones es, de nuevo, llenar la jarra 3 litros, obteniendo nuevamente el nodo (3,0). ¿Qué implica? Un bucle. Por tanto, en la búsqueda en profundidad debemos tener cuidado, pues si no existe una lista de cambios a lo largo del árbol se puede acabar traduciendo en un bucle infinito.

ESTRATEGIAS PARA LA BUSQUEDA DESINFORMADA:

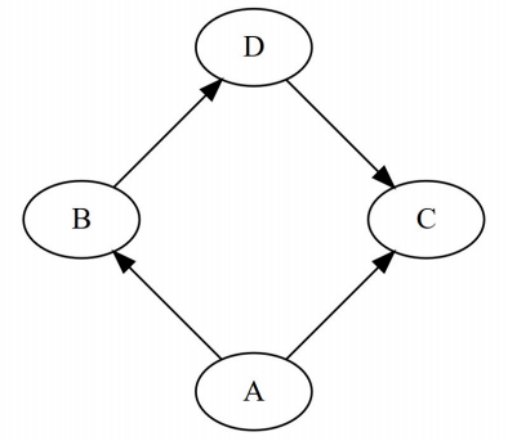
BUSQUEDA DE COSTE UNIFORME

Cuando los costes son iguales la búsqueda en anchura es óptima ya que siempre expande el nodo menos profundo que no ha sido expandido. En la búsqueda uniforme, en vez de expandir el nodo menos profundo, se expande el nodo cuyo camino que hemos tenido que hacer para llegar a él es el de coste más bajo. Esto se consigue gestionando la lista de ABIERTOS como una lista de prioridad ordenada por el coste de los caminos.

El algoritmo es completo puesto que el camino a la solución o meta tiene un coste y como examinamos sistemáticamente los caminos por su coste entonces existe un camino a la solución, por lo que tarde o temprano llegaremos a la solución. A diferencia de la búsqueda en anchura y la búsqueda en profundidad, los bucles no importan ya que por cada iteración de ese bucle el coste del camino se va haciendo más grande, por lo que más pronto que tarde saldremos de él. El algoritmo también es óptimo porque debido al funcionamiento del mismo siempre llegaremos a la solución de menor coste primero.

En cuanto a las complejidades en este caso son iguales $O(b^{1+\left\lceil \frac{C^*}{\epsilon} \right\rceil})$ donde C* indica el coste a la solución óptima y donde toda acción tiene un coste al menos igual (o superior) que €.

A continuación vamos a comparar la búsqueda de coste uniforme y la búsqueda en anchura. Supongamos que disponemos del siguiente grafo y tenemos el nodo A como nodo inicial y el nodo D como nodo meta.



Realizamos la búsqueda en anchura como hemos procedido anteriormente:

ACTUAL	SUCESOS	ABIERTO
		A
A	BC	BC
B	DA	CDA
C	DA	DADA

Ahora si realizamos la búsqueda de coste uniforme y aplicamos los siguientes coste a cada arista veamos lo que sucede: A → B = 1, A → C = 3, B → D = 1, D → C = 2.

ACTUAL	SUCESOS	ABIERTO
		A(0)
A(0)	B(1)C(3)	B(1)C(3)
B(1)	D(2)A(2)	D(2)A(2)C(3)
D	[…]	[…]

ESTRATEGIAS PARA LA BUSQUEDA DESINFORMADA:

BUSQUEDA EN PROFUNDIDAD LIMITADA

El inconveniente que tiene la búsqueda en profundidad puede ser resuelto mediante la búsqueda en profundidad limitada estableciendo un límite de profundidad L. Esto es, aquellos nodos que se encuentren en la profundidad L serán tratados como si no tuvieran sucesores.

La búsqueda en profundidad no es un algoritmo completo puesto que si elegimos la profundidad L < d entonces estamos diciendo que la solución menos profunda está por debajo del límite impuesto, por lo que nunca llegaremos a ella. Por otro lado tampoco es un algoritmo optimo, si L > d ocurriría lo mismo que con la búsqueda en profundidad normal. En cuanto a la complejidad temporal es del valor de $O(r^{p+1})$ y la complejidad espacial $O(r * p)$.

En conclusión, la búsqueda en profundidad limitada presenta la ventaja del económico ahorro en memoria, pues la búsqueda está limitada. Sin embargo, el gran inconveniente es la incompletitud, esto es, por un lado, al no saber a qué nivel se haya la primera solución puede que tardemos en encontrarla e incluso, en el peor de los casos, que ni siquiera la encontremos, pues la profundidad es arbitraria.

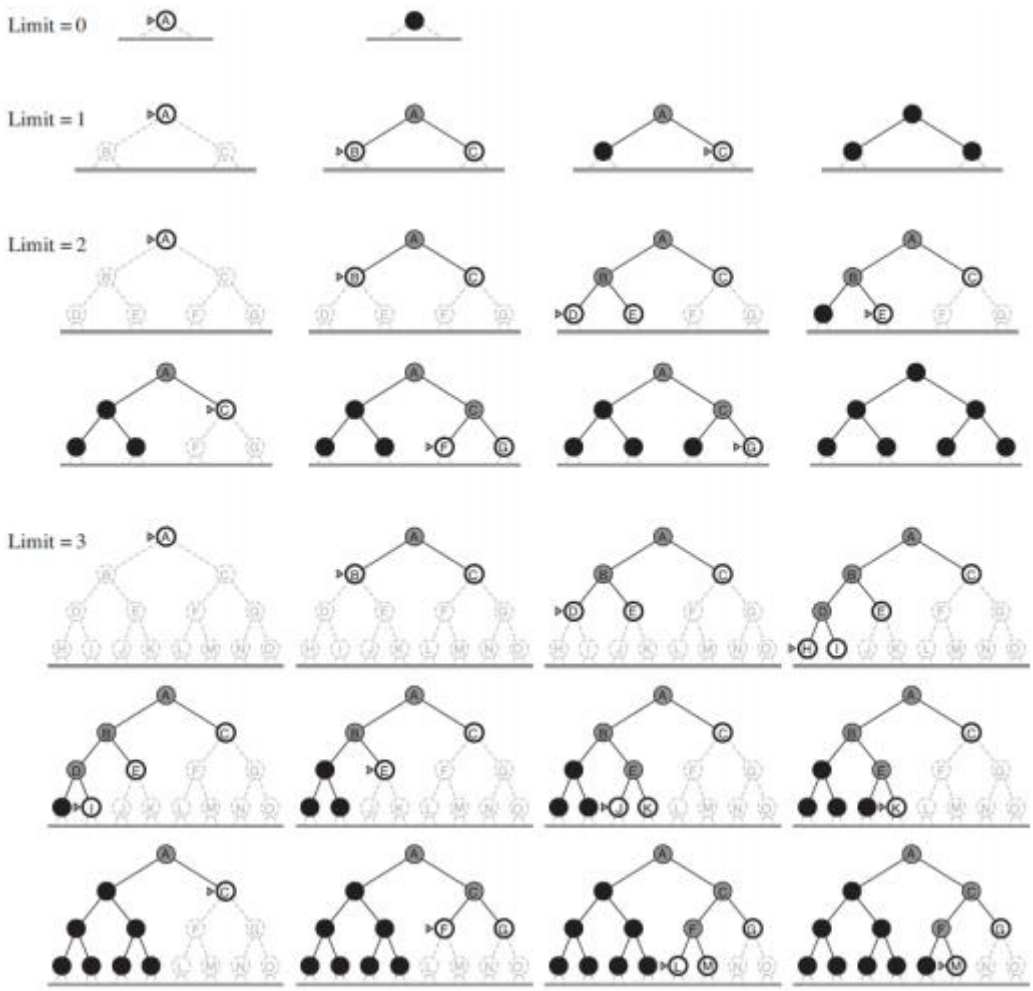
ESTRATEGIAS PARA LA BUSQUEDA DESINFORMADA:

BUSQUEDA EN PROFUNDIDAD ITERATIVA

Una posible alternativa a la búsqueda en profundidad limitada es ir aumentando, de forma gradual, la profundidad, es decir, empezamos desde el nivel 0, si no hallamos la primera solución aumentamos un nivel; y si no encontramos una primera solución, aumentamos de nuevo la profundidad en uno, y así sucesivamente, hasta alcanzar el nivel d máximo de profundidad del árbol. La búsqueda de profundidad iterativa puede parecer derrochadora, porque los estados se generan múltiples veces. Pero esto no es muy costoso. La razón es que en un árbol de búsqueda con el mismo (o casi el mismo) factor de ramificación en cada nivel, la mayor parte de los nodos está en el nivel inferior, entonces no importa mucho que los niveles superiores se generen múltiples veces.

En cuanto a la optimalidad del algoritmo, la búsqueda en profundidad iterativa será optima si los costes para ir de nodo a nodo son todos iguales. El algoritmo además será completo si el factor de ramificación es finito.

En cuanto a las complejidades, la complejidad temporal vendrá del número de nodos generados para encontrar la meta $O(b^d)$. La complejidad espacial será $O(b * d)$.



ESTRATEGIAS PARA LA BUSQUEDA DESINFORMADA:

BUSQUEDA BIDIRECCIONAL

En la búsqueda bidireccional básicamente se ejecutan dos búsquedas de forma simultánea, una hacia adelante desde el estado inicial y otra hacia atrás desde el estado meta, creyendo que ambas se encontraran en algún punto en el interior.

El algoritmo de búsqueda bidireccional si es completo cuando en ambas direcciones se usa búsqueda en profundidad y en factor de ramificación es finito. Por otro lado también es óptimo si los costes de los pasos son todos idénticos. En cuanto a las complejidades, tanto la complejidad espacial y la complejidad es igual del orden de $O(b^{d/2})$.

COMPARACION ENTRE BUSQUEDAS

En la siguiente tabla se compararán los diferentes algoritmos entre sí:

Criterio	Primero en anchura	Costo uniforme	Primero en profundidad	Profundidad limitada	Profundidad iterativa	Bidireccional (si aplicable)
¿Completa?	Sí[a]	Sí[a,b]	No	No	Sí[a]	Sí[a,d]
Tiempo	$O(b^{d+1})$	$O(b^{\lceil C^*/\epsilon \rceil})$	$O(b^m)$	$O(b^\ell)$	$O(b^d)$	$O(b^{d/2})$
Espacio	$O(b^{d+1})$	$O(b^{\lceil C^*/\epsilon \rceil})$	$O(bm)$	$O(b\ell)$	$O(bd)$	$O(b^{d/2})$
¿Optimal?	Sí[c]	Sí	No	No	Sí[c]	Sí[c,d]

Figura 3.17 Evaluación de estrategias de búsqueda. b es el factor de ramificación; d es la profundidad de la solución más superficial; m es la máxima profundidad del árbol de búsqueda; ℓ es el límite de profundidad. Los superíndice significan lo siguiente: [a] completa si b es finita; [b] completa si los costos son $\geq \epsilon$ para ϵ positivo; [c] optimal si los costos son iguales; [d] si en ambas direcciones se utiliza la búsqueda primero en anchura.

EJERCICIOS

Para los siguientes problemas interprete, abstraiga y formalice (es decir, describa formalmente en términos computables):

(a) alguna formulación del espacio de estados.

(b) descripción de los estados iniciales.

(c) funciones u operadores de transición entre estados (o acciones de un agente que cambien el estado).

(d) función que establece el coste del paso de unos estados a otros.

(e) conjunto de estados meta o solución y criterio que permite valorarlos como tales. Cuando se pueda, presente varias formulaciones alternativas.

1.- Problema de las jarras: Dadas dos jarras de tres y de cuatro litros de capacidad, sin ninguna graduación, averiguar cómo llegar a tener en una de ellas exactamente dos litros sin usar medidas de capacidad auxiliares.

En primer lugar vamos a formular un espacio de estados, en este caso serán dos variables, una variable X que determinara la jarra1 y una variable Y que determinar la jarra2. Los posibles valores de dichas variables son:

X = 1, 2, 3, 4

Y = 1, 2, 3

En cuanto a los estados, se determinarán mediante dichas variables de la siguiente manera (X,Y) de forma que el estado inicial será (0,0) indicando que ambas jarras están vacías.

En cuanto a los operadores, tendremos cuatro operadores básicos y dos operadores más para pasar de un estado a otro y son los siguientes:

OP1 (X,Y) → (4,Y): llenamos la jarra de 4L

OP2 (X,Y) → (X,3): llenamos la jarra de 3L

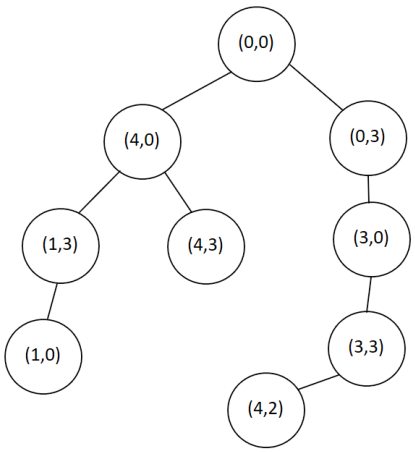
OP3 (X,Y) → (0,Y): vaciamos la jarra de 4L

OP4 (X,Y) → (X,0): vaciamos la jarra de 3L

OP5(X,Y) → (0, X + Y): vertemos la jarra de 4L en la jarra de 3L

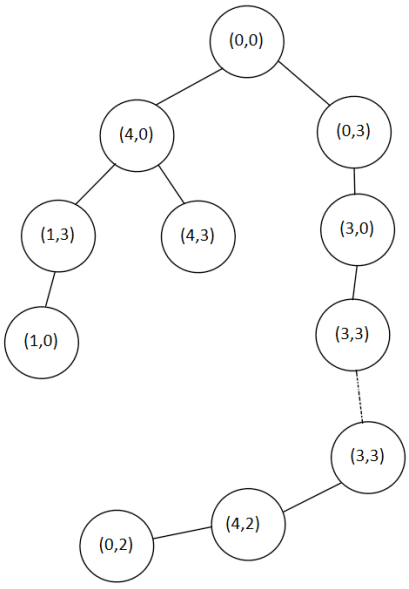
OP6(X,Y) → (X + Y, 0): vertemos la jarra de 3L en la jarra de 4L

Como en este caso no hay coste para transaccionar entre estados, no podemos definir una función de coste.



En cuanto al grafo del problema, algunos estados no se han incluido pues ya estaban repetidos, esto se hace para evitar la aparición de bucles.

En cuanto al método de búsqueda podemos elegir la búsqueda en anchura o búsqueda en profundidad, pero en realidad el método de búsqueda más eficiente seria búsqueda bidireccional pues conocemos tanto el nodo inicial como el nodo meta y las complejidades espacial y temporal son mucho menores $O(b^{d/2})$.



Aplicamos la búsqueda bidireccional partiendo del nodo inicial (0,0) y teniendo como nodo meta (0,2). En el momento en el que lleguemos al mismo nodo, en este caso el nodo (3,3), hemos encontrado la solución.

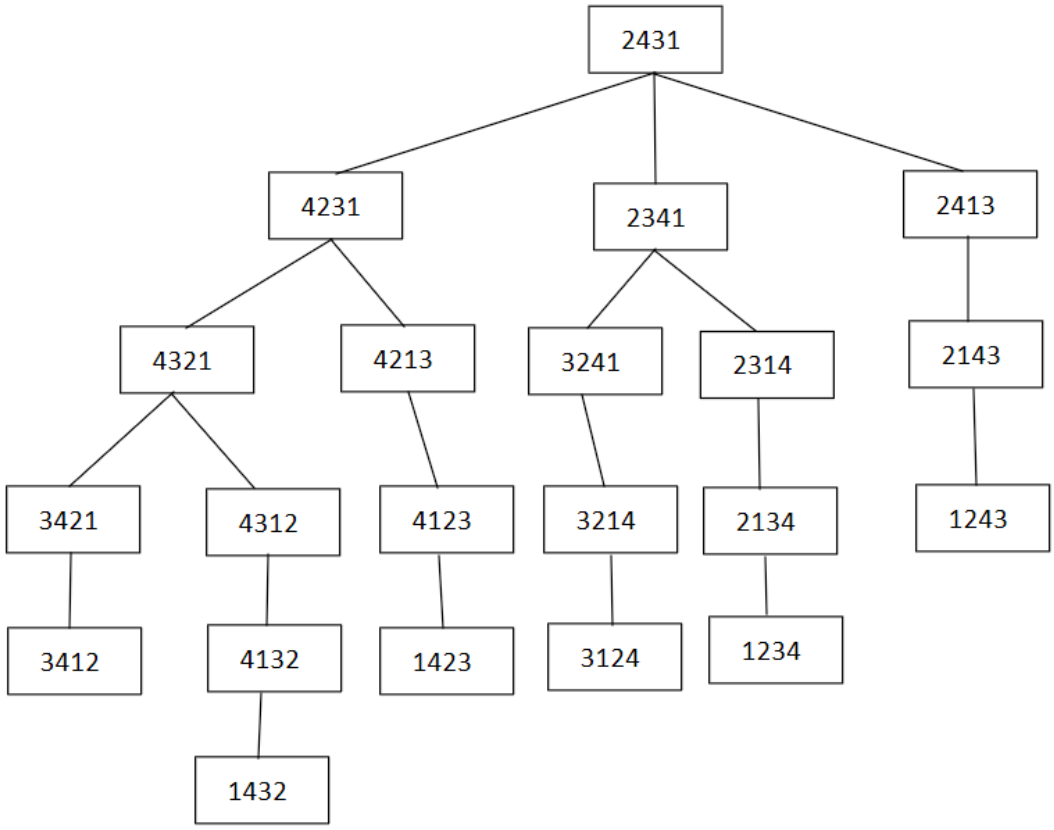
2.- Problema del bastidor: En un bastidor hay cuatro fichas móviles numeradas, colocadas de la siguiente forma 2431. El objetivo es mover las fichas de tal forma que queden ordenadas por su valor numérico, mediante movimientos de intercambio entre fichas contiguas.

Disponemos de un nodo inicial que es el 2431 y un nodo meta que es 1234. Además disponemos de operaciones de cambio de estados:

OP1: Intercambiar el primer elemento por el segundo.

OP2: Intercambiar el segundo elemento por el tercero.

OP3: intercambiar el tercer elemento por el cuarto.



A continuación construimos el grafo de estados y vemos con que algoritmo lo podríamos resolver.

En primer lugar realizare los cuatro primeros pasos del algoritmo de búsqueda en profundidad para saber su funcionamiento:

Actual	Sucesores	Abiertos
		2431
2431	4231, 2341, 2413	4231, 2341, 2413
4231	4321, 4213	4321, 4213, 2341, 2413
4321	3421, 4312	3421, 4312, 4213, 2341, 2413
3421	3412	3412, 4312, 4213, 2341, 2413

Una vez realizados los cuatro primeros pasos del algoritmo de búsqueda en profundidad, realizo al completo el algoritmo de búsqueda en anchura.

Actual	Sucesores	Abiertos
		2431
2431	4231, 2341, 2413	4231, 2341, 2413
4231	4321, 4213	2341, 2413, 4321, 4213
2341	3241, 2314	2413, 4321, 4213, 3241, 2314
2413	2143	4321, 4213, 3241, 2314, 2143
4321	3421, 4312	4213, 3241, 2314, 2143, 3421, 4312
4213	4123	3241, 2314, 2143, 3421, 4312, 4123
3241	3214	2314, 2143, 3421, 4312, 4123, 3214
2314	2134	2143, 3421, 4312, 4123, 3214, 2134
2143	1243	3421, 4312, 4123, 3214, 2134, 1243
3421	3412	4312, 4123, 3214, 2134, 1243, 3412
4312	4132	4123, 3214, 2134, 1243, 3412, 4132
4123	1423	3214, 2134, 1243, 3412, 4132, 1423
3214	3124	2134, 1243, 3412, 4132, 1423, 3124
2134	1234	1243, 3412, 4132, 1423, 3124, 1234
1243	-	3412, 4132, 1423, 3124, 1234
3412	-	4132, 1423, 3124, 1234
4132	1432	1423, 3124, 1234, 1432
1423	-	3124, 1234, 1432
3124	-	1234, 1432
1234		

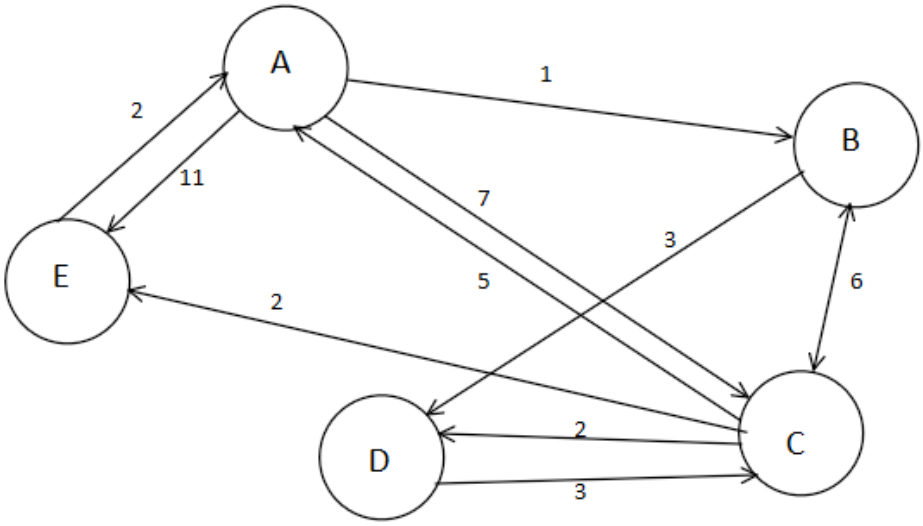
3.- Problema de un examen: En una red de metro están incluidas cinco estaciones A, B, C, D, E unidas por distintas líneas descritas por la matriz de adyacencia ponderada.

	A	B	C	D	E
A	0	1	7		11
B		0	6	3	
C	5	6	0	2	2
D			3	0	
E	2				0

Resolver metódicamente el problema de ir desde A hasta E por el metodo de búsqueda optimal o de coste uniforme.

El primer paso es definir tanto el estado inicial, como el estado meta. El objetivo es ir a la estación E partiendo desde la estación A, es obvio que el estado inicial será A y el estado meta será E. Dada la matriz de adyacencia las celdas no enumeradas indican que el camino a través de ellas es imposible.

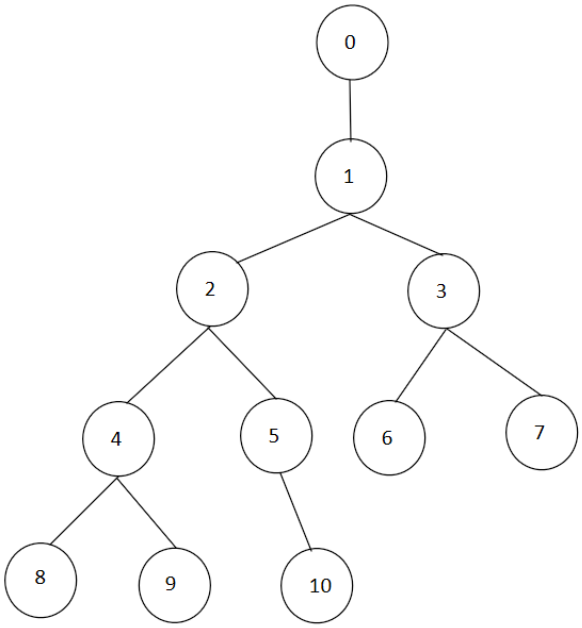
A continuación se presenta el grafo del problema y se procederá a resolverlo mediante el algoritmo indicado.



Actual	Sucesores	Abiertos
		A
A	AE(11), AB(1), AC(7)	AB(1), AC(7), AE(11)
AB	ABC(6 + 1), ABD(3 + 1)	ABD(3 + 1), ABC(6 + 1), AC(7), AE(11)
ABD	ABDC(4 + 3)	ABDC(4 + 3), ABC(7), AC(7), AE(11)
ABDC	ABDCE(7 + 2)	ABC(7), AC(7), ABDCE(7 + 2), AE(11)
ABC	ABCE(7 + 2), ABCD(7 + 2)	AC(7), ABDCE(9), ABCE(9), ABCD(7 + 2), AE(11)
AC	ACE(7 + 2), ACB(7 + 6), ACD(7 + 2)	ABDCE(9), ABCE(9), ABCD(9), ACE(7 + 2), ACD(7 + 2), AE(11), ACB(7 + 6),
ABDCE		

4.- Problema de un examen: Sea un espacio de estados en el que el primero de ellos es el número 0 y donde para cada estado hay dos sucesores dados por los operadores O1 y O2 definidos por $O1(n) = 2n$ y $O2(n) = 2n + 1$. Supongamos que el estado meta es el 10.

a) Construir el grafo y el árbol de búsqueda. (b) describir como seria visitados los nodos en la búsqueda en anchura. (c) resolverlo por algún metodo. (d) Razonar si puede resolverse por búsqueda en profundidad.

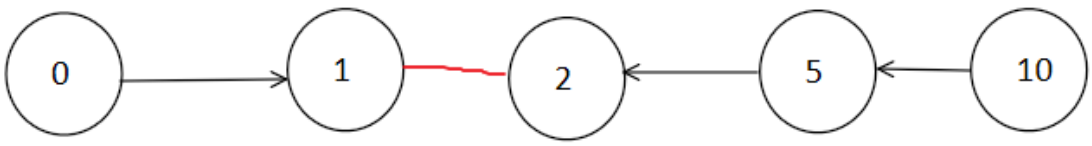


En cuanto a cómo serían los nodos visitados si aplicásemos la búsqueda en profundidad. Los nodos más cercanos al nodo raíz serían los primero en ser visitados de forma que para hallar la solución en este problema deberíamos recorrer los 10 nodos hasta llegar el nodo deseado.

A continuación resolveré el problema aplicando la búsqueda en profundidad:

Actual	Sucesor	Abiertos
		0
0	1	1
1	2, 3	2, 3
2	4, 5	4, 5, 3
4	8, 9	8, 9, 5, 3
8	-	9, 5, 3
9	-	5, 3
5	10	10, 3
10		

En cuanto a si es posible resolverlo de mediante la búsqueda bidireccional, en la búsqueda bidireccional realizaríamos dos búsquedas simultaneas, una desde el nodo raíz y otra desde el nodo meta. Si aplicamos operaciones dese el nodo raíz podemos llegar a un punto de encuentro entre ambas búsquedas. Obviamente solo podemos aplicar aquellas operaciones que sean validas, es decir, solo aquellas operaciones que nos devuelvan como resultado un número entero. Desde el nodo meta podemos aplicar O1 y O2 de forma alterna hasta llegar a un nodo de encuentro en cuyo caso en este problema sería el nodo2.

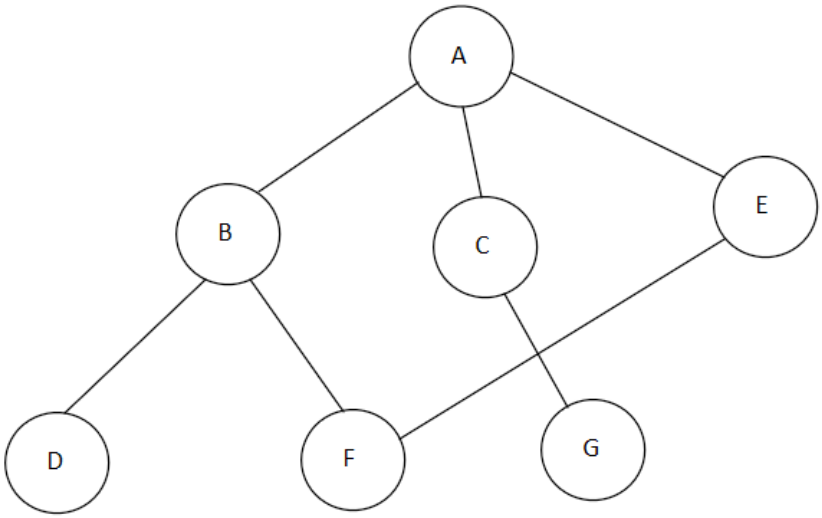


5.- Cuestión de un examen: Comprobar que la búsqueda en anchura en un caso particular de la búsqueda optimal.

Lo que si es cierto es que si en la búsqueda optimal establecemos que todos los costes de las aristas del grafo son iguales y positivos, estamos convirtiendo la búsqueda optimal en una búsqueda en anchura y por lo tanto, la búsqueda en anchura si es un caso particular de la búsqueda optimal.

6.- Cuestión de un examen: Comprobar mediante ejemplos que las búsquedas en profundidad y en profundidad limitada pueden fallar, es decir, no encontrar ninguna meta, aunque las haya accesibles desde un estado inicial.

Veamos un ejemplo de búsqueda en profundidad:

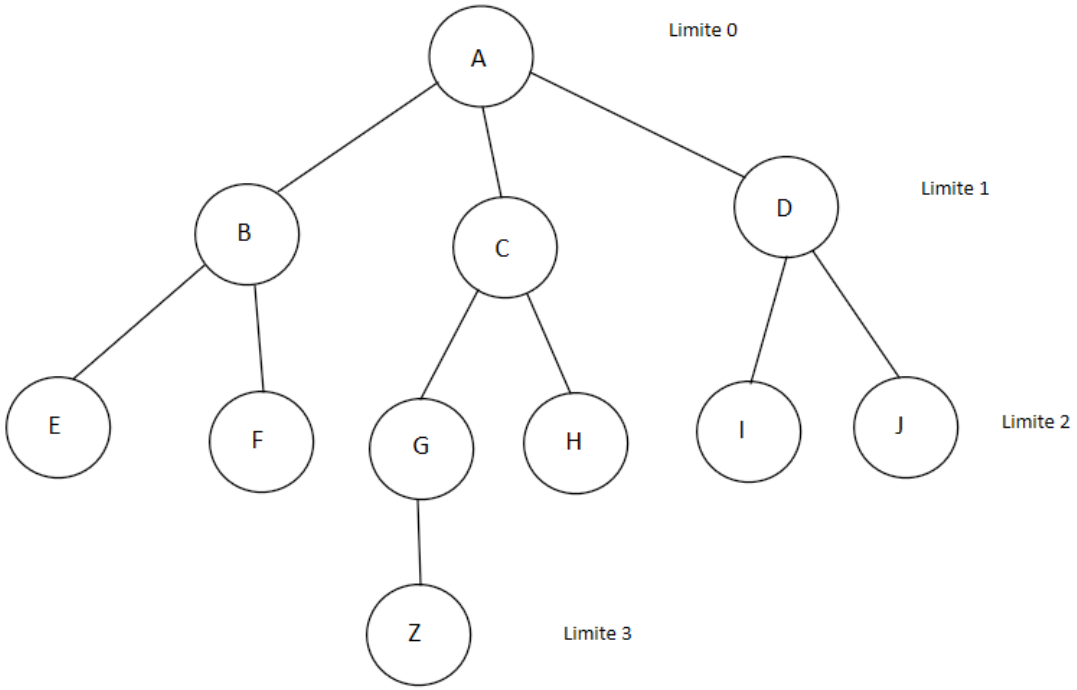


Partiendo de la base de que partimos desde el nodo A como el nodo inicial y tenemos al nodo G como nodo meta.

Actual	Sucesor	Abierto
		A
A	B, C, E	B, C, E
B	D, F	D, F, C, E
D	-	F, C, E
F	E	E, C, E
E	F	F, E, C, E

Como podemos observar, estaríamos todo el rato en bucle sin llegar nunca a encontrar la solución, de esta forma podemos concluir que la búsqueda en profundidad no es completa.

Vamos a ver ahora un ejemplo de la búsqueda limitada. El problema de la búsqueda limitada es que si establecemos un cierto límite n y la solución se encuentra en n + 1, nunca encontraremos la solución.



Imaginemos que tenemos como nodo inicial el nodo A y como nodo meta el nodo Z. En cuanto al límite, pongo el imite en n = 2:

Actual	Sucesor	Abierto
		A
A	B, C, D	B, C, D
B	E, F	E, F, C, D
E	-	F, C, D
F	-	C, D
C	G, H	G, H, D
G	-	H, D
H	-	D
D	I, J	I, J
I	-	J
J	-	-

Como podemos observar no hemos llegado a la solución ya que estaba a un nivel de profundidad mas bajo del que hemos establecido por lo que podemos concluir que la búsqueda en profundidad limitada también es incompleta.

7.- Cuestión de un examen: Comprobar que las búsqueda actual y optimal son óptimas, es decir, que siempre encuentran primero el mejor camino.

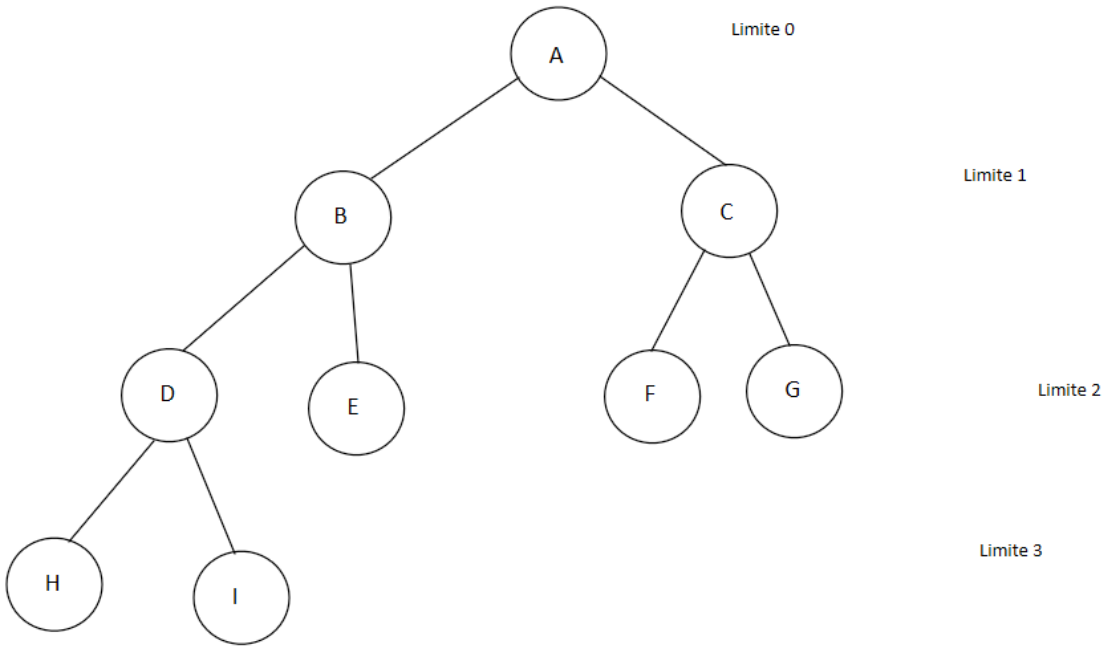
En el caso de la búsqueda en anchura, al realizar la búsqueda exhaustiva en cada nivel nunca se dará el caso de que la solución que encuentre no sea optima ya que al recorrer nivel por

nivel el grafo, la solución encontrada siempre será la mas cercana a la raíz y por lo tanto la mas optima.

En el caso de la búsqueda optimal siempre encuentra la solución optima ya que al realizar el algoritmo estamos buscando los caminos de menor coste desde la raíz hasta el nodo solución. Al buscar los caminos de menor coste, el camino hacia la solución siempre será el camino de menor coste del grafo.

8.- Cuestión de un examen: Comprobar que si $L_{i+1} = L_i + 1$ para cada i, también la búsqueda en profundidad iterativa es óptima. En otro caso puede ser subóptima, es decir, puede encontrar una solución con un camino cuya longitud difiera de la del optimo como máximo en $L_{i+1} - L_i$.

En primer lugar hay que decir que $L_{i+1} = L_i + 1$ significa que vamos a ir aumentando el nivel de búsqueda de uno en uno. Vamos a ver un ejemplo:



Siendo el nodo inicial el nodo A y siendo el nodo meta el nodo H, vamos a aplicar la búsqueda en profundidad limitada primero hasta el límite 1 e iremos aumentando de niveles de forma progresiva:

Actual	Sucesor	Abierto
		A
A	B, C	B, C
B	-	C
C	-	-

No hemos encontrado la solución, por ello aumentamos en 1 el límite:

Actual	Sucesor	Abierto
		A
A	B, C	B, C
B	D, E	D, E, C
D	-	E, C
E	-	C
C	F, G	F, G
F	-	G
G	-	-

No hemos encontrado la solución, por ello aumentamos el límite hasta 3:

Actual	Sucesor	Abierto
		A
A	B, C	B, C
B	D, E	D, E, C
D	H, I	H, I, E, C
H		

Como vemos si aumentamos de uno en uno el límite, la búsqueda es óptima ya que se encuentra la solución en el menor número de pasos.

9.- Cuestión de un examen: Describa que es una búsqueda en profundidad, cuales son sus variantes y porque se desarrollan estas.

BUSQUEDA INFORMADA(HEURISTICA)

En el tema anterior la búsqueda se realizaba sistemáticamente explorando l0os nodos del grafo pero sin emplear ninguna información disponible sobre la preferencia de unos caminos o de otros o de la proximidad de las metas en las partes de los caminos aun por recorrer.

Cuando disponemos de algún tipo de información se habla de heurística y los métodos correspondientes son los métodos heurísticos. En estos métodos se utiliza la información disponible para elegir el desarrollo de unos estados frente a otros, tomando para desarrollar de la lista de abiertos aquellos nodos cuya distancia con el nodo meta sea la menor.

Veamos un ejemplo: Imaginemos que queremos resolver el problema de 8-puzzle donde disponemos de un estado inicial y 8n estado final que se muestran a continuación.

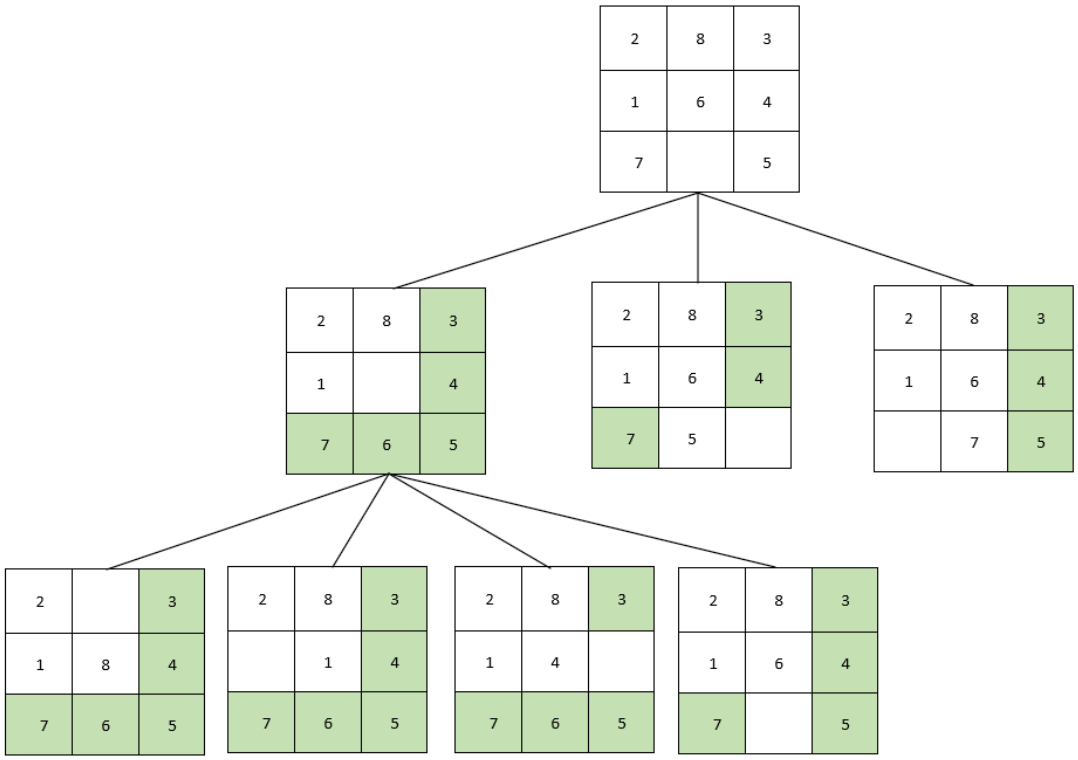
Estado inicial:

2	8	3
1	6	4
7		5

Estado meta:

1	2	3
8		4
7	6	5

Los estados representan cada uno de los posibles tableros, para cambiar de un estado a otro disponemos de cuatro movimientos. Podemos mover una ficha hacia arriba, abajo, izquierda y derecha. Podemos representar el problema mediante el siguiente grafo:



Como podemos observar en este caso estamos trabajando con información adicional, las casillas marcadas en verde son las casillas que concuerdan con el estado meta. Por tanto aquellos nodos con las casillas marcadas en verde serán nodos que más se acerquen al nodo meta, estos nodos serán los que exploraremos primero.

En resumen, podríamos usar un criterio de número de piezas descolocadas como indicador de las distancias que falta para llegar a la meta y realizar exploraciones de aquellos nodos cuyo número de piezas descolocadas sea menor, por lo tanto a medida que vayamos desarrollando el árbol los sucesores de dichos nodos tendrán cada vez menor número de piezas descolocadas. Realmente al aplicar la información disponible en el problema estamos aplicando una heurística que es en este caso el número de piezas descolocadas.

Vamos a explicar un poco mejor lo que hemos hecho: El número de piezas descolocadas es una estimación de los pasos que faltan por realizar hasta llegar al nodo meta. Esta estimación es una **función heurística** h: {estados} → [0, +∞]. Esta función heurística nos da una estimación de cuanto nos queda para llegar al estado final h(estado) ≥ 0 y h(meta) = 0.

Esta estimación es de **información incierta y puede conducir a engaño**. Al usarla se sacrifica certeza para ganar economía, "podando" el árbol de búsqueda para centrarse en las ramas más prometedoras. **Se combina la heurística**(futuro) **con el gasto realizado**(pasado) para establecer métodos seguros y económicos que combinan ambas ventajas.

Si tuviésemos una función que nos dijese el valor exacto que queda para llegar a la meta, solo habría que seguirla para llegar al nodo final. Esta función se denomina función heurística perfecta ya que estima la distancia exacta de cada estado hasta la meta y se denota con h*. Aunque no dispongamos de h* es muy útil comparar con ella, por ejemplo, en el ejercicio anterior

h(estado) = número de fichas descolocadas,

0 ≤ h(estado) ≤ h*(estados).

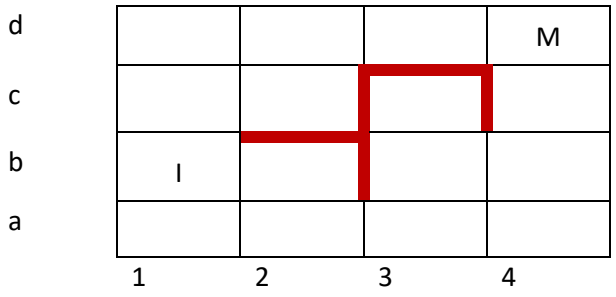
Cuanto más cerca este nuestra función de h* más información nos proporcionara.

Los métodos que utilizan cierta información heurística que nos estiman el mejor camino hacia la meta se denomina búsqueda heurística o búsqueda de primero el mejor. La idea es la siguiente:

1. Tenemos una función de evaluación heurística 'f' que nos ayuda a decidir qué nodo es el mejor para expandir(f(n) será menor en los nodo prometedores).
2. Se expandirá el nodo 'n' para el cual se obtenga menor f(n).
3. Se terminará el proceso cuando el nodo 'n' a expandir sea el nodo meta.

En el ejercicio de 8-puzzle f(n) sería el número de piezas descolocadas → f(n) = h(n). En algunas funciones podemos introducir más valores para mejorar la heurística o evitar que esta sea demasiado optimista, como por ejemplo, incluir el nivel de profundidad en el que se encuentra el nodo f(n) = g(n) + h(n).

Veamos otro problema. Disponemos del siguiente plano donde se indica el punto inicial desde donde partimos y el punto final. Como podemos observar hay zonas por donde no podemos pasar, marcadas en color rojo.



El cálculo de la distancia desde cada casilla a la meta será estimado. Si no hubiese obstáculos habría varias soluciones con camino mínimo. Ahora debemos modelizar el problema, el estado del tablero seria cada uno de los nodos mientras que los posibles movimientos serían las operaciones posibles para pasar de un estado a otro. Dichos movimientos serian: abajo, izquierda, derecha y arriba.

Una heurística posible podría ser emplear la distancia de Manhattan($d = |x - x1| + |y - y1|$) para calcular la distancia mínima entre cada casilla con el punto final. Por ejemplo:

Estado	h(estado) Manhattan	h(estado) letras	h*(estado)
a1	6	3	6
a2	5	3	5
a3	4	…	4
a4	3	…	3
b1	5	…	5
b2	6	…	6
b3	3	…	3
b4	2	…	2
c1	…	4	…
c2	…	3	…
c3	…	…	…
…	…	…	…

Podemos emplear varias heurísticas distintas, otra seria por ejemplo la distancia euclídea. Pero sin ninguna duda la que más información nos proporciona en este caso es la distancia de Manhattan.

METODOS VORACES:
BUSQUEDA EN ESCALADA

Esta búsqueda siempre se mueve hacia el valor creciente, es decir, al nodo cuyo valor sea mejor que el nodo actual. Funciona con un solo estado actual y generalmente se mueve solo a los vecinos del estado. Este tipo de búsqueda se caracteriza porque usa poca memoria o nula, y porque encuentra solución en grandes estados donde los algoritmos sistemáticos son inadecuados. El algoritmo consiste en lo siguiente:

1. Empezar con que la lista Actual es una lista formada únicamente por el estado inicial.
2. Hasta que Actual = meta o no haya cambios en Actual, hacer:
   2.1. Presentar la lista Actual si su estado es meta.
   2.2. Tomar los sucesores de Actual y usar la función heurística para puntuar cada uno de ellos.
   2.3. Si uno de los sucesores tiene mejor puntuación que Actual, hacer Actual igual a la lista anterior aumentada en el sucesor.
3. Presentar fallo y terminar.

Este algoritmo no es completo, ni optimo ya que puede ocurrir que exista un valor heurístico más bajo pero que no sea el valor heurístico más bajo global. Además al mirar solo estados vecinos puede ocurrir que el valor heurístico no sea mejorado por lo que el algoritmo se quedaría atascado. Otra opción que puede suceder es que todos los estados vecinos tengan el mismo valor heurístico y por lo tanto el algoritmo sea incapaz de encontrar el camino. Una opción para mejorar esto sería permitir un movimiento lateral con el fin de salir de este estado.

Para entender mejor el algoritmo vamos a ver un ejemplo sencillo:

Estados	(h(estados)
I	4
A	3
D	5
E	6

Con este algoritmo elegiríamos directamente la A.

METODOS VORACES:
PRIMERO EL MEJOR

La búsqueda voraz primero el mejor trata de expandir el nodo más cercano al objetivo. Así, evalúa los nodos utilizando solamente la función heurística: f(n) = h(n). Funciona de la misma forma que la búsqueda optimal gestionando la lista de abiertos como una cola de prioridad, usando solo la función heurística para ordenarla y priorizarla, examinando aquellos nodos cuyo valor sea el menor. El algoritmo es el siguiente:

1. Abiertos: lista formada por el estado inicial.
2. Mientras que la lista de abiertos no esté vacía:
   2.1. Eliminar de la lista aquel nodo cuyo valor sea el menor y añadirlo a actual.
   2.2. Si el nodo de la lista actual es nodo meta, presentar su lista y terminar.
   2.3. Si no es meta, tomar los sucesores de dichos nodos y usar la función heurística para puntuar cada uno de ellos, incorporando la lista resultado de ampliar actual con dichos nodos a la lista de abiertos en su orden.
3. Preguntar si hay fallo y terminar.

Con una buena función heurística, la búsqueda de primero el mejor puede mejorar el coste computacional. Lo que no queda garantizado es que el camino sea óptimo. Además debemos tener en cuenta el coste de calcular la función heurística en cada nodo.

A continuación vamos a ver cómo trabaja el algoritmo mediante un ejemplo. El objetivo es encontrar una ruta en Rumania, con el objetivo de llenar a Bucarest. Para ello conocemos las distancias en línea recta desde cualquier ciudad de Rumania hasta Bucarest.

Arad	366	**Mehadia**	241
Bucarest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Dobreta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374

Figura 4.1 Valores de h_{DLR}. Distancias en línea recta a Bucarest.

El primer nodo a expandir desde Arad será Sibiu, porque está más cerca de Bucarest que Zerind o que Timisoara. El siguiente nodo a expandir será Fagaras, porque es la más cercana. Fagaras en su turno genera Bucarest, que es el objetivo. Sin embargo, no es óptimo: el camino vía Sibiu y Fagaras a Bucarest es 32 kilómetros más largo que el camino por Rimnicu Vilcea y Pitesti. Esto muestra por qué se llama algoritmo «avaro» (en cada paso trata de ponerse tan cerca del objetivo como pueda).

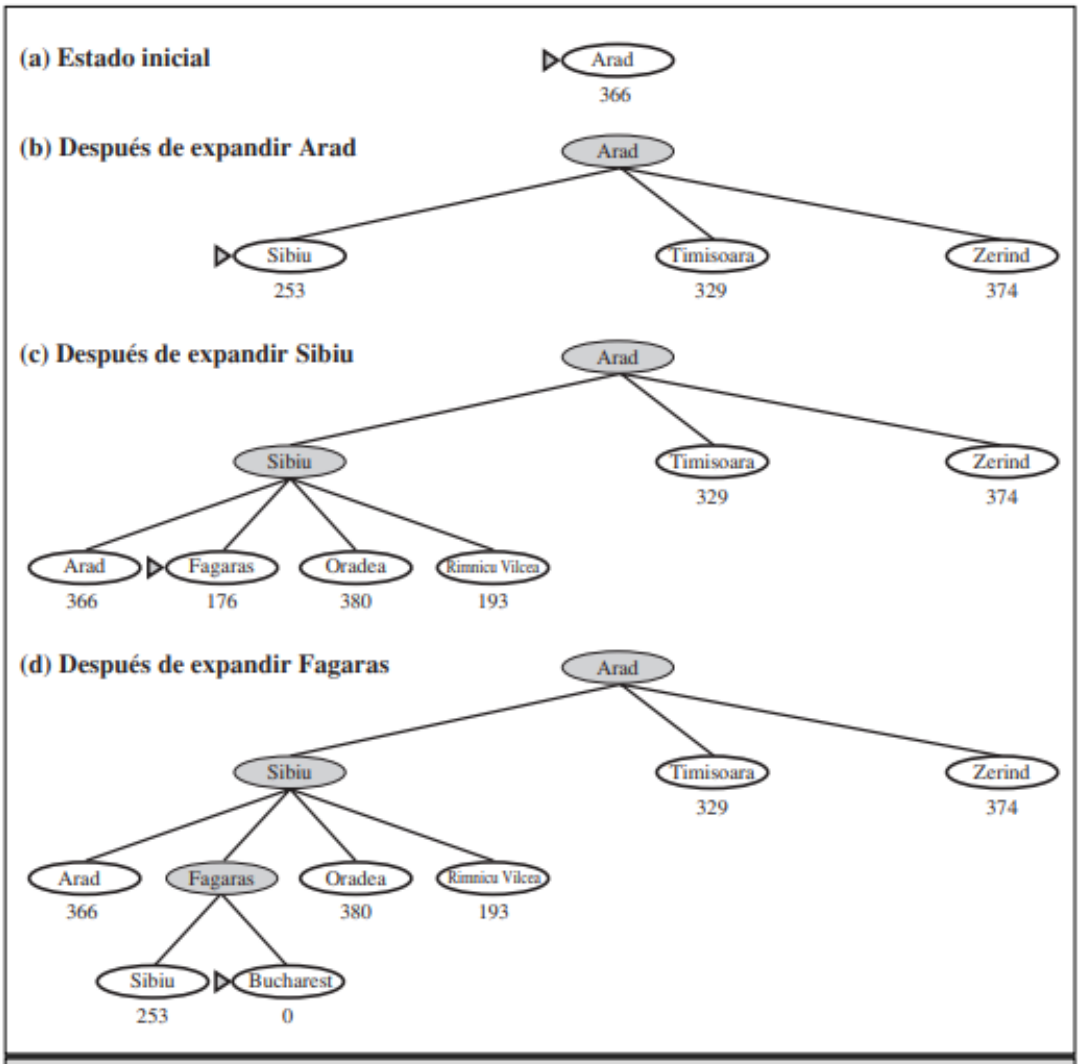


Figura 4.2 Etapas en una búsqueda primero el mejor avara para Bucarest utilizando la heurística distancia en línea recta h_{DLR}. Etiquetamos lo nodos con sus h-valores.

La búsqueda voraz primero el mejor se parece a la búsqueda primero en profundidad en el modo que prefiere seguir un camino hacia el objetivo, pero volverá atrás cuando llegue a un callejón sin salida. Sufre los mismos defectos que la búsqueda primero en profundidad, no es óptima, y es incompleta (porque puede ir hacia abajo en un camino infinito y nunca volver para intentar otras posibilidades). La complejidad en tiempo y espacio, del caso peor, es $O(b^m)$, donde m es la profundidad máxima del espacio de búsqueda. Con una buena función, sin embargo, pueden reducir la complejidad considerablemente. La cantidad de la reducción depende del problema particular y de la calidad de la heurística.

BUSQUEDA A*

La búsqueda A* es una variante de la búsqueda primero el mejor. Esta búsqueda evalúa los nodos combinando g(n), es decir, el coste para alcanzar el nodo, y h(n), el coste de ir al nodo objetivo: $f(n) = g(n) + h(n)$

Ya que g(n) determina el coste de ir desde el nodo inicial hasta el nodo n, y h(n) el coste estimado del camino más barato desde el nodo n hasta el nodo meta, tenemos que:

$$f(n) = coste\ mas\ barato\ estimado\ de\ la\ solucion\ a\ traves\ de\ n$$

Si la heurística es adecuada con este metodo se consigue un algoritmo completo, optimo y con un coste computacional menor. La lista de abiertos se ordena según el valor de f(n) de menor a mayor, eligiendo el menor para desarrollar en la siguiente iteración. Otra opción es hacer la media de g(n) + h(n) y ordenarlos de menor a mayor según ese valor.

Para garantizar que A* encuentra siempre el camino de mínimo coste tanto el grafo generado como la función h deben satisfacer las siguiente condiciones:

1. Cada nodo del grafo tiene un numero finito de sucesores(en caso de que los hubiera).
2. El coste de cada arco del grafo deberá ser mayor que una cierta cantidad positiva denotada con ð.
3. La función h deberá satisfacer la siguiente condicion: En todos los nodos del grafo de búsqueda se deberá cumplir que el coste heurístico h deberá ser menor que el coste ideal h*, h(n) ≤ h*(n).

Denotando como h* la heurística ideal, es decir, aquella que proporciona la información perfecta sobre el coste optimo desde cada nodo hasta la meta. Siendo además $f*(n) = g(n) + h*(n)$ el coste de la mejor solución que pase por el nodo n.

Dada una función heurística puede ocurrir que:

1. h(n) = 0 para todo valor de n, diríamos que la búsqueda no nos estaría proporcionando información por lo que A* se convertiría en una búsqueda de coste uniforme.
2. h(n) > h*(n) para algún valor de n, es entonces cuando A* puede resultar no optimo pues estaríamos diciendo que la función h(n) es capaz de sobreestimar el valor de h*(n). Tendríamos lo que se denomina como un estimador optimista.
3. h(n) = h*(n) para cada valor de n, entonces no habría problema ya que en cada caso sabríamos en todo momento que dirección habría que tomar para llegar a la meta.
4. h(n) ≤ h*(n) para cada valor de n, es entonces cuando A* encontrara el camino optimo a la solución. Resultará un algoritmo completo, optimo y con coste computacional menor cuando h(n) se aproxime más al valor de h(n).

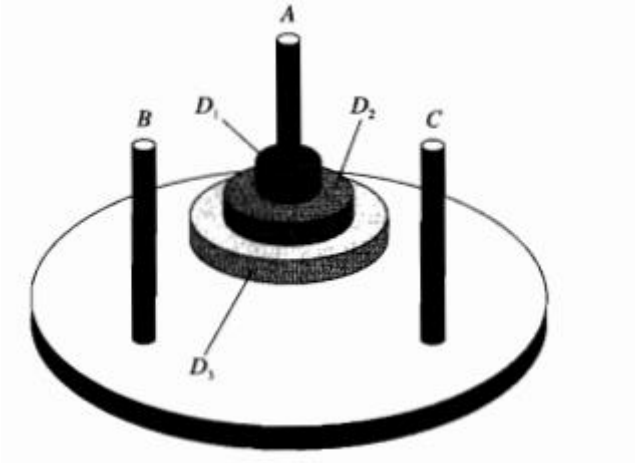
Además se dice que una heurística h(n) es más informativa que otra heurística h' si cumple que 0 ≤ h'(n) ≤ h(n) ≤ h*(n) para cada n. Es decir, si la búsqueda heurística h se aproxima en mayor medida al coste ideal h* en contraposición con h', se dice entonces que la búsqueda es más informativa que la otra. Cuanto más informativa sea una búsqueda menos nodos habrá que explorar hasta llegar al nodo meta. Por otro lado, se dice que una heurística es consistente cuando dados dos nodos ni y nj, donde nj es el sucesor de ni, la diferencia entre ambos valores heurísticos es menor o igual que la distancia entre ambos nodos. Heurística consistente → heurística admisible.

$$h(ni) - h(nj) \leq c(ni, nj)$$

EJERCICIOS

1.- Problema de las torres de Hanoi: Se dispone de un tablero con tres barras verticales. En una de ella hay insertadas k fichas circulares perforadas, con tamaños decrecientes desde la de abajo hasta la de arriba. El problema consiste en cambiar las fichas de barra, moviéndolas de una en una, desde una barra a otra, de forma que en cada caso se mueva sola la de arriba de una de las barras y que nuca quede una de menor tamaño debajo de otra mayor. (a) Formalizar adecuadamente le problema; (b) Diseñar una heurística admisible; (c) En el caso de k = 3 determine los cuatro primeros estados de la lista de ABIERTOS al resolver por el metodo de primero el mejor; (d) En el caso de k = 3 resolver metódicamente con A*.

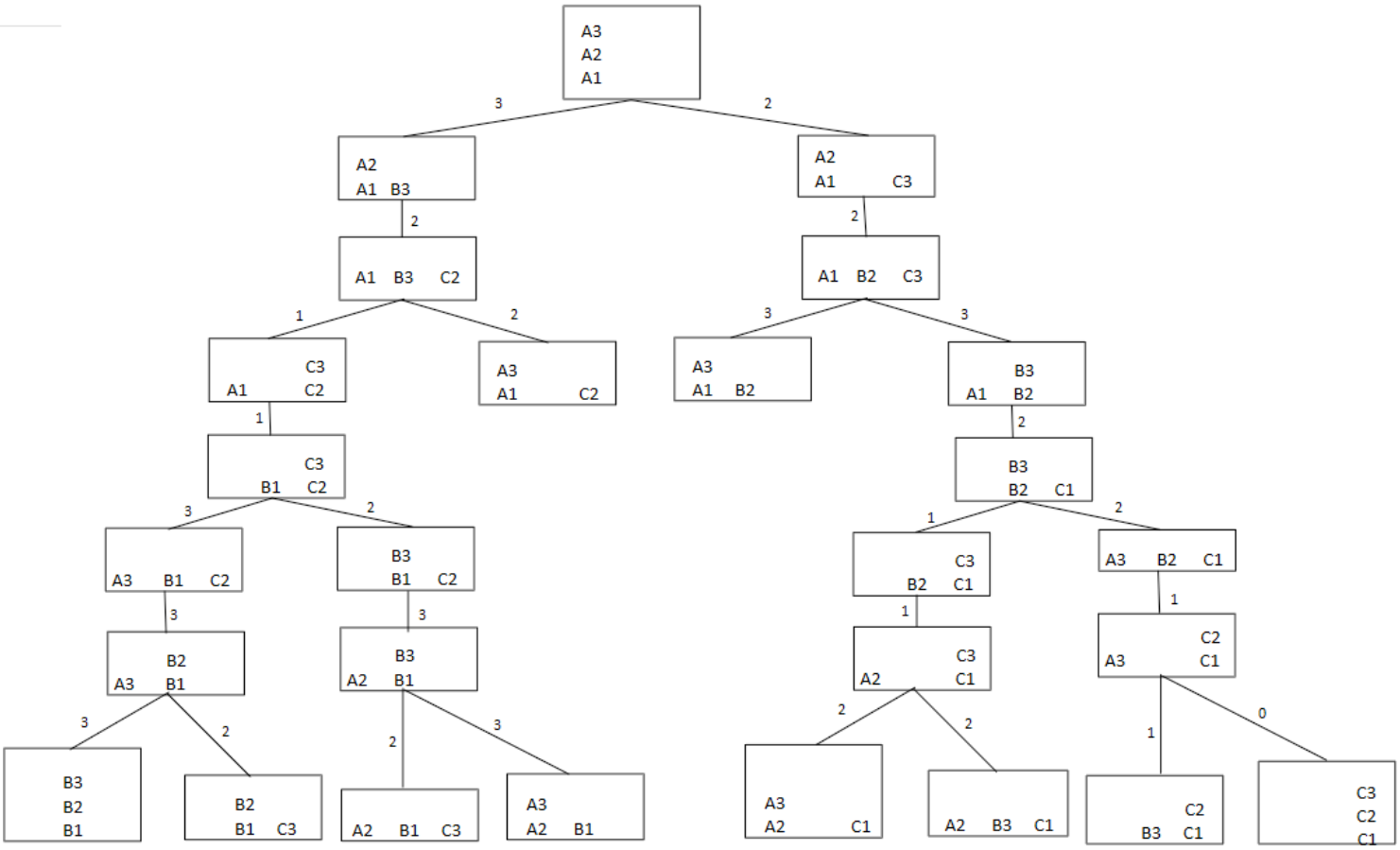
En el problema tenemos varillas a las que denominaremos A, B, C y en las que pueden colocarse discos. Los discos se nombraras según su tamaño y según en la varilla en la que se encuentren siendo A1 el disco mas grande y situado en la varilla A. Inicialmente los discos están colocados de forma ordenada en una de las varillas. Supongamos que los discos están situados en la varilla A, entonces el estado inicial de nuestro problema seria A1, A2, A3. El objetivo es colocar los discos de la varilla A en la varilla C respetando el orden en el que están colocados.



De todos los discos que podamos mover siempre moveremos el mas grande, pero no lo moveremos si con esos deshacemos el movimiento que acabamos de realizar. Siempre que sea posible los movimientos impares los realizaremos hacia la derecha y los movimientos pares hacia la izquierda. Cuando no sea posible realizar los movimientos según este esquema, moveremos el disco en la otra dirección y después retomaremos el esquema definido.

En cuanto a la heurística creare una puntuación de acuerdo a la diferencia entre la ubicación actual del disco y la del estado final, de forma que aquella situaciones en las que estén los tres discos descolocados se les dará una puntuación de 3, mientras que en aquellas situaciones en las que haya algún disco colocado en su posición, se les reducirá dicha puntuación.

Construyo el grafo del problema:



Comienzo realizando las cuatro primeras iteraciones dl algoritmo de primero el mejor:

Actual	Sucesor	Abierto
		A1A2A3
A1A2A3	A1A2B3, A1A2C3	A1A2C3, A1A2B3
A1A2C3	A1B2C3	A1B2C3, A1A2B3
A1B2C3	A1A3B2, A1B2B3	A1A3B2, A1B2B3, A1A2B3
A1A3B2	-	A1B2B3, A1A2B3
A1B2B3	……	……

Ahora procedo a resolver el problema mediante la búsqueda A*:

Actual	Sucesor	Abierto
		A1A2A3
A1A2A3	A1A2B3(3), A1A2C3(2)	A1A2C3(2), A1A2B3(3)
A1A2C3	A1B2C3(4)	A1A2B3(3), A1B2C3(4)
A1A2B3	A1B3C2(5)	A1B2C3(4), A1B3C2(5)
A1B2C3	A1A3B2(7), A1B2B3(7)	A1B3C2(5), A1A3B2(7), A1B2B3(7)
A1B3C2	A1C2C3(6), A1A3C2(7)	A1C2C3(6), A1A3B2(7), A1B2B3(7) , A1A3C2(7)
A1C2C3	B1C2C3(7)	A1A3B2(7), A1B2B3(7) , A1A3C2(7), B1C2C3(7)
A1A3B2	-	A1B2B3(7) , A1A3C2(7), B1C2C3(7)

A1B2B3	B2B3C1(9)	A1A3C2(7), B1C2C3(7), B2B3C1(9)
A1A3C2	-	B1C2C3(7), B2B3C1(9)
B1C2C3	A3B1C2(10), B1B3C2(9)	B2B3C1(9), B1B3C2(9), A3B1C2(10)
B2B3C1	B2C1C3(10), A3B2C1(11)	B1B3C2(9), A3B1C2(10), B2C1C3(10), A3B2C1(11)
B1B3C2	A2B1B3(12)	A3B1C2(10), B2C1C3(10), A3B2C1(11), A2B1B3(12)
A3B1C2	A3B1B2(13)	B2C1C3(10), A3B2C1(11), A2B1B3(12), A3B1B2(13)
B2C1C3	A2C1C3(11)	A3B2C1(11), A2C1C3(11), A2B1B3(12), A3B1B2(13)
A3B2C1	A3C1C2(12)	A2C1C3(11), A2B1B3(12), A3C1C2(12), A3B1B2(13)
A2C1C3	A2A3C1(13), A2B3C1(13)	A2B1B3(12), A3C1C2(12), A3B1B2(13), A2A3C1(13), A2B3C1(13)
A2B1B3	A2B1C3(14), A2A3B1(15)	A3C1C2(12), A3B1B2(13), A2A3C1(13), A2B3C1(13), A2B1C3(14), A2A3B1(15)
A3C1C2	B3C1C2(13), C1C2C3(12)	C1C2C3(12), A3B1B2(13), A2A3C1(13), B3C1C2(13), A2B3C1(13), A2B1C3(14), A2A3B1(15)
C1C2C3		

Una vez terminado el algoritmo podemos determinar que la ruta para lograr el nodo final es la siguiente: A1A2A3, A1A2C3, A1B2C3, A1B2B3, B2B3C1, A3B2C1, A3C1C2, C1C2C3 con un coste total de 15.

2.- Problema de la urbanización: En una urbanización de trazado rectangular hay cinco calles de norte a sur que se cruzan con cuatro avistas de este a oeste, delimitando doce parcelas rectangulares iguales, con doble longitud que anchura, cada una de las cuales está ocupada por cuatro casas, con entrada por los chaflanes. Si alguien, que conoce la planta de la urbanización pero no las calles intransitables por las obras, quiere ir del cruce de la primera calle y avenida a la casa situada en el cruce de la última calle con la segunda avenida, ignorando que estén cortados de la tercera calle y el tercer tramo de la segunda avenida. (a) ¿Qué metodo de búsqueda puede usar?; (b) Indicar una heurística admisible y resolver el problema metódicamente con una búsqueda heurística.

En primer lugar voy a crear una representación del mapa para poder entender mejor el enunciado.

				Fin
Inicio				

Las zonas coloreadas de rojo son zonas cortadas por las que no se puede pasar. En cuanto a los movimientos, los únicos que se pueden realizar son movimientos horizontales y verticales. En cuanto a los estados, son cada una de las casillas.

En cuanto al método de búsqueda podemos aplicar A* o primero el mejor. En cuanto a la heurística, para resolver el problema lo mas optimo posible lo ideal seria calcular la distancia que hay desde cada celda a la celda meta. Para calcular dicha distancia emplearemos la distancia de Manhattan desde cada celda hasta la celda final:

$$d(x,y) = \sqrt{(x1 - x2) + (y1 - y2)}$$

El problema de la heurística de Manhattan es que no tiene en cuenta los obstáculos por lo que en este caso no nos serviría. Por ello lo mas optimo es realizar el calculo de cada celda mediante una heurística propia adaptada al problema. Para poder entender mejor la resolución de algoritmo enumerare cada celda de forma que la celda inicial será a1 la celda final era b5.

d					
c					
b					Fin
a	Inicio				
	1	2	3	4	5

Actual	Sucesor	Abierto
		a1(9)
a1	b1(8), a2(10)	b1(8), a2(10)
b1	c1(7), b2(9)	c1(7), b2(9), a2(10)
c1	d1(6)	d1(6), b2(9), a2(10)
d1	d2(5)	d2(5), b2(9), a2(10)
d2	d3(4)	d3(4), b2(9), a2(10)
d3	c3(3), d4(3)	c3(3), d4(3), b2(9), a2(10)
c3	c4(2)	c4(2), d4(3), b2(9), a2(10)
c4	b4(1), c5(1)	b4(1), c5(1), d4(3), b2(9), a2(10)
b4	b5(0)	b5(0), c5(1), d4(3), b2(9), a2(10)
b5		

3.- Cuestión de examen: En bajo qué condiciones es completo el método de búsqueda A*.

El algoritmo A* será optimo cuando apliquemos una heurística adecuada, es equivalente en la practica una disminución del factor de ramificación. Por otro lado para garantizar que una heurística sea adecuada debemos tener en consideración lo siguiente:

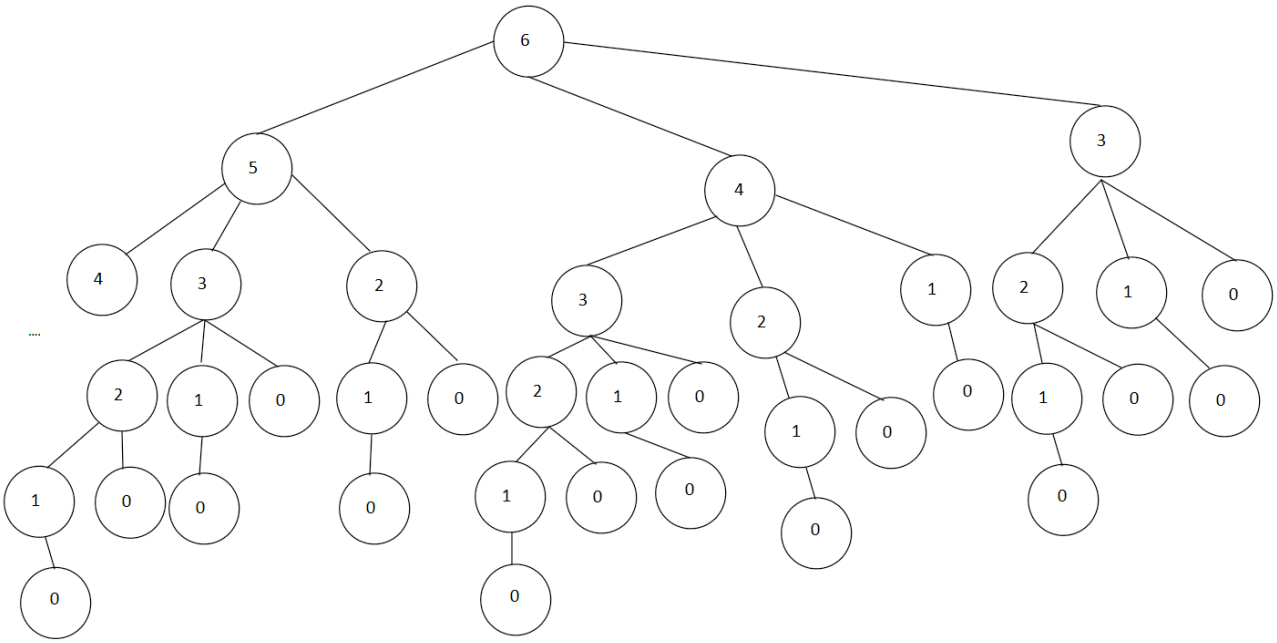
1. Cada nodo del grafo tiene un numero finito de sucesores, en caso de que los tuviese.
2. El coste en cada arco del grafo deberá ser mayor que una cierta cantidad positiva
3. La función heurística h(n) deberá satisfacer la siguiente condición en todos los nodos del grafo de búsqueda, se deberá cumplir que el coste heurístico h es menor que el coste idea, es decir, $h^* \leq h(n) \leq h^*(n)$.

Denotando como h* la heurística ideal, es decir, la que proporciona la información perfecta sobre el coste optimo por el mejor camino desde cada nodo hasta la meta más cercana, siendo además $f^*(n) = g(n) + h^*(n)$ el coste de la mejor solución que pase por n.

BUSQUEDA CON ADVERSARIOS (INTRODUCCIÓN A JUEGOS)

Hasta ahora hemos desarrollado programas para solucionar problemas que en su naturaleza son estáticos. En este tema vamos a tratar de resolver problemas dinámicos, es decir, modificando la solución del problema en el transcurso del mismo.

Para comenzar vamos a ir viendo el juego del Nim, en el que existe una bolsa de fichas con *n* fichas iguales. Hay dos jugadores y cada uno podrá quitar fichas del montón(1, 2 o 3 al mismo tiempo). Los turnos entre los dos jugadores se van alternando y pierde aquel que retire la última ficha.



En cada iteración vamos poniendo las fichas que quedan en el montón. El jugador1 puede obtener un +1 o un -1, si el segundo jugador saca la última ficha el primer jugador obtiene una ganancia de +1.

Con esa puntuación en cada nodo podemos decir que se elija la rama más favorable para el jugador1 o para el jugador2 en cada caso. Entonces de arriba a abajo vamos subiendo e indicando en cada nodo la puntuación potencial. Esta transición se realiza de la siguiente manera:

- En nodos de nivel par(jugador1) se transmite el máximo(puntuación de los nodos desde la hoja hasta la raíz).
- En nodos de nivel impar(jugador2) se transmite el mínimo(puntuación de los nodos desde la hoja hasta la raíz).

Marcamos las puntuaciones en las hojas dependiendo de quien haya ganado(jugador1 = +1, jugador2 = -1). Una vez tenemos esas puntuaciones vamos subiendo con el algoritmo MinMax, es decir, cuando le toque al jugador1(Max) miraremos sus hijos y cogeremos la máxima puntuación mientras que si el jugador2(Min) miraremos sus hijos y cogeremos la mínima puntuación.

Una vez disponemos de todas las puntuaciones, cogemos la etiqueta de la raíz y seguimos el árbol con aquellos nodos que tengan la misma etiqueta. En este caso, el mejor camino para que gane el jugador1 es 6, 5, 4, 1, 0 o 6, 5, 3, 1, 0.

Como hemos visto nosotros nos centramos en juegos de dos jugadores que juegan alternamente, con un numero infinito de opciones y con competencia completa. En el ejemplo que hemos visto hemos tratado el juego como una búsqueda donde tenemos dos jugadores(Max y Min), unos estados(situaciones de juegos), un estado inicial(comienzo), un estado final, unos movimientos y un premio. Todo esto lo hemos representado en un árbol donde cada nodo son los turnos y los arcos son las jugadas. Además cada turno corresponde a un nivel de profundidad del árbol, los niveles pares al 1 y los impares al 2.

Algunas mejoras que podríamos aplicar son las siguientes:

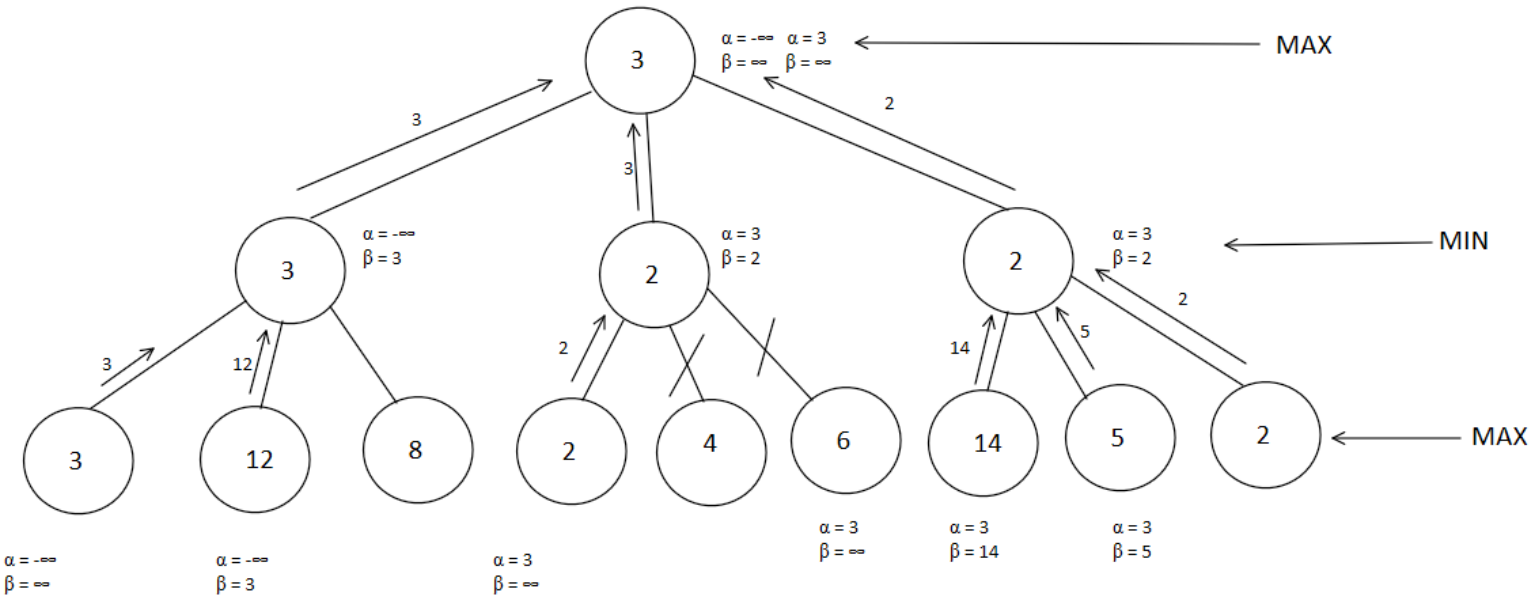
1. Desarrollar el árbol solo hasta un determinado nivel de profundidad “horizonte” y evaluar los nodos de dicho horizonte mediante funciones heurísticas que evalúen la juagada.
2. Realizar la poda de subramas innecesarias para evaluar los nodos.

MINMAX CON PODA ALPHA-BETA

El algoritmo es el siguiente:

- Si el nodo N es un nodo hoja, devolver V(N, α, β) = valor de N.
- Si N es un nodo MAX y no nodo hoja, para cada hijo n, α = max(m, V(N, α, β)):
  - Si α ≥ β devolver β como V(N, α, β).
  - Si usamos todos los hijos , devolver α como V(N, α, β).
- Si N es un nodo MIN y no nodo hoja, para cada hijo n, β min(m, V(N, α, β)):
  - Si α ≥ β devolver α como V(N, α, β).
  - Si usados todos los hijos, devolver β como V(N, α, β).

Veamos el siguiente ejemplo:



Aplicamos el metodo poda α-β de izquierda a derecha, siendo el primer nodo el jugador max. Inicialmente el nodo raíz α = -∞ y β = ∞. Comenzamos desde la raíz y como vamos de izquierda a derecha bajamos α y β hasta el nodo 3 directamente pues no tiene "valores".

Estamos en un nodo hoja, V(3, -∞, ∞) = 3, subimos el 3. Ahora el 3 está en un nodo MIN:

$$\beta = \min\big(m, V(N, \alpha, \beta)\big) = \min(3, +\infty) = 3$$

Cogemos el siguiente hijo, en este caso el nodo hoja 12:

$V(12, -\infty, 3) = 12$ subimos el 12 → m = 12

Ahora estamos en un nodo Min $\beta = \min(12,3) = 3$ como α no es ≥ β seguimos explorando el siguiente hijo.

Cogemos el siguiente hijo, en este caso el nodo hoja 8:

$V(8, -\infty, 3) = 8$ subimos el 8 → m = 8

Ahora estamos en un nodo Min $\beta = \min(8,3) = 3$ como α no es ≥ β seguimos explorando el siguiente hijo. Como en este caso ya no quedan más hijos subimos β como $V(N, \alpha, \beta)$.

Ahora estamos en un nodo Max con m = 3 $\alpha = \max(3, -\infty)$, actualizamos α en el nodo raíz. Por lo que en la raíz nos quedaría (3, +∞).

Exploramos el segundo hijo de la raíz. En este caso el nodo hoja 2:

$V(2, 3, \infty) = 2$ subimos el 2 → m = 2

Ahora estamos en un nodo Min $\beta = \min(2, \infty) = 2$ actualizamos β (β = 2 y α = 3). Como en este caso α si es ≥ β devolvemos α hacia Max y podamos el resto de hijos.

Ahora estemos en un nodo Max con m = 3, $\alpha = \max(3,3) = 3$ se queda igual.

Exploramos el tercer hijo de la raíz. En este caso el nodo hoja 14: m = 14.

Ahora estamos en un nodo Min $\beta = \min(14, \infty) = 14$ actualizamos β (α = 3, β = 14). Como en este caso α no es ≥ β exploramos el siguiente hijo.

Cogemos el siguiente hijo, en este caso el nodo hoja 5 → m = 5.

Ahora estamos en un nodo Mi $\beta = \min(5, 14) = 5$n actualizamos β (α = 3, β = 5). Como en este caso α no es ≥ β exploramos el siguiente hijo.

Cogemos el siguiente hijo, en este caso el nodo hoja 2 → m = 2.

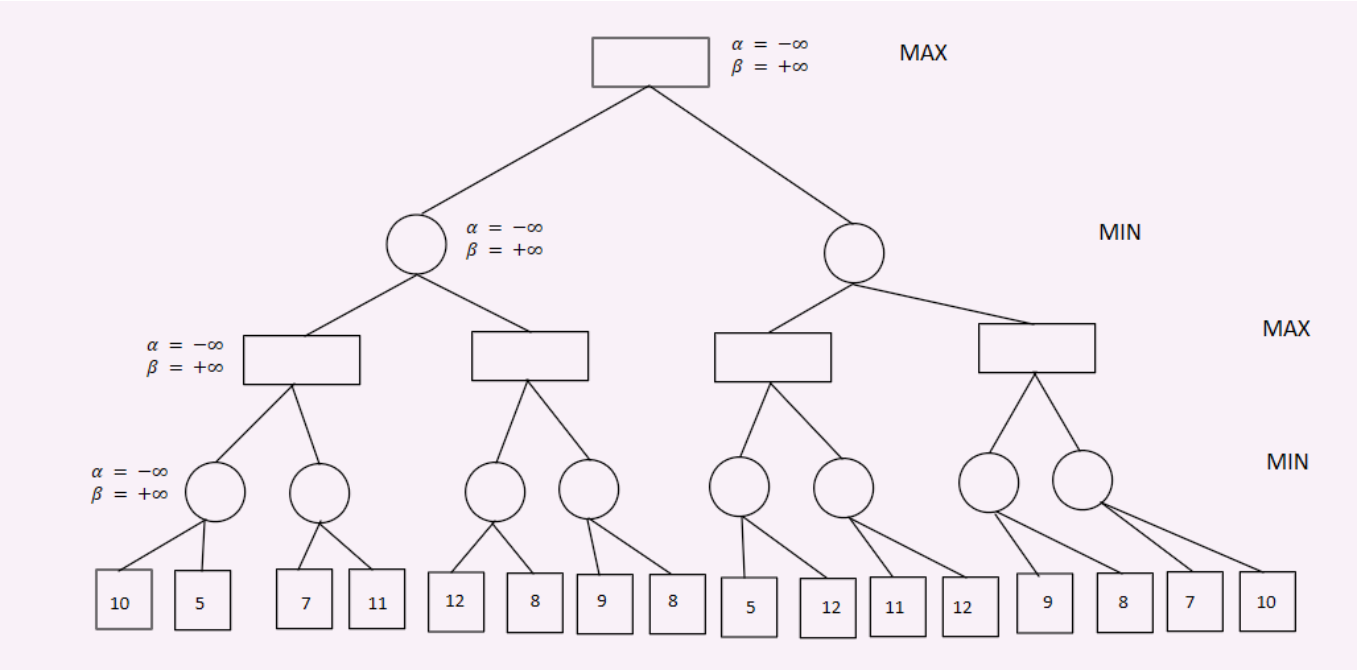
Ahora estamos en un nodo Min $\beta = \min(2,2) = 2$ actualizamos β (α = 3, β = 2). Como en este caso α no es ≥ β exploramos el siguiente hijo. Como ya hemos explorado todos los hijos, subimos β.

Ahora estamos en un nodo Max $\alpha = \max(2,3) = 3$ se queda igual. Por lo tanto al final nos queda que la raíz es (3, +∞).

Con esta poda Alpha-beta lo que hemos hecho es quitarnos ramas que no nos interesan. En cada nodo hemos ido etiquetando con los valores que iba devolviendo el algoritmo. El etiquetado realizado con la poda coincide con el MinMax únicamente en el nodo raíz.

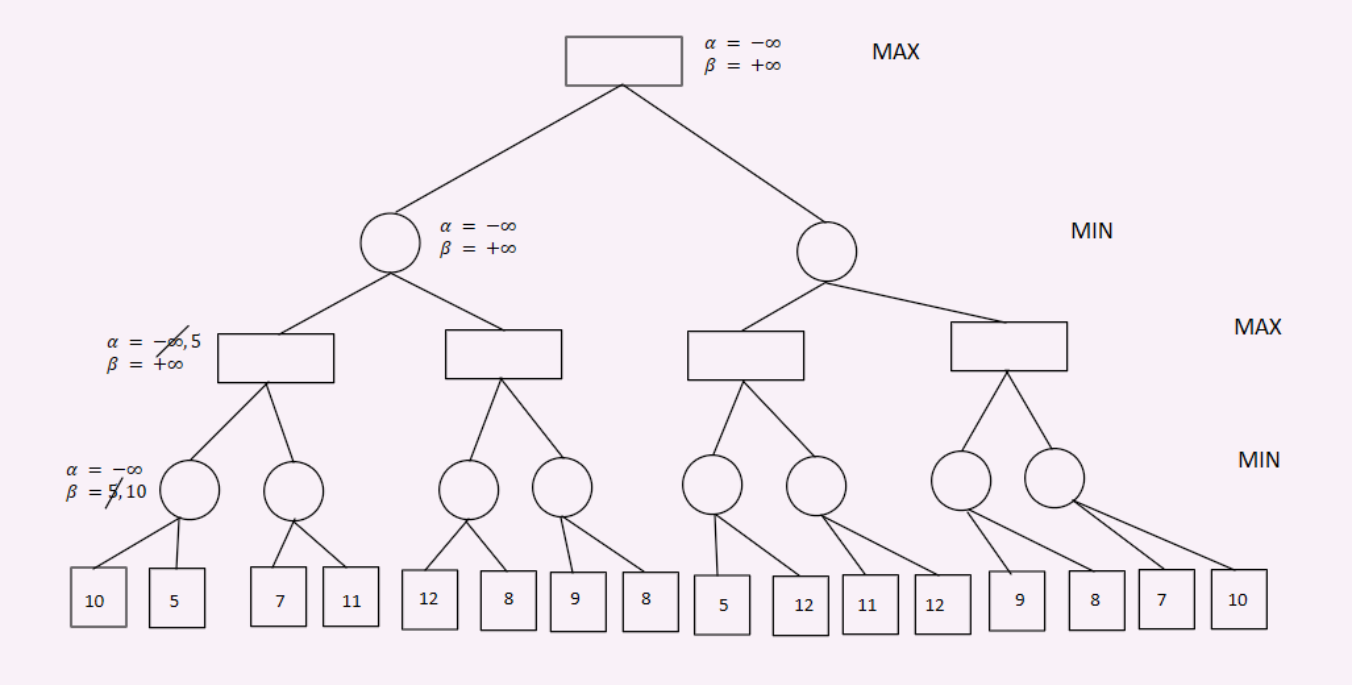
EJERCICIOS

1.- Ejercicio internet: Resolver el grafo aplicando el algoritmo de poda Alpha-beta.

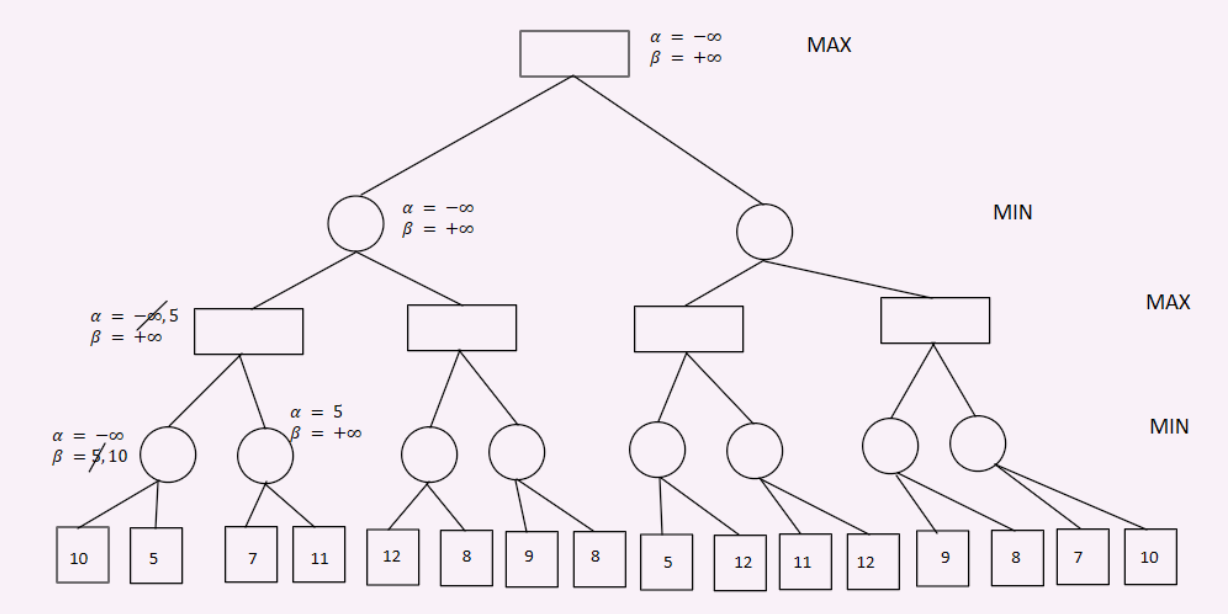


Comenzamos aplicando el algoritmo desde el nodo hoja mas a la izquierda posible. El nodo que queremos calcular es un nodo min, es decir, vamos a modificar el valor de β. Aplicamos la siguiente formula $\beta = \min(10, +\infty) = 10$. Modificamos el valor de β y pasamos al siguiente nodo hijo. Ahora aplicamos el mismo procedimiento que antes $\beta = \min(5, 10) = 5$.

Una vez hemos calculado el valor del nodo lo subimos al nodo padre, dicho nodo es un nodo maximizador por lo que debemos modificar el valor de α esta vez. $\alpha = \max(5, -\infty) = 5$.

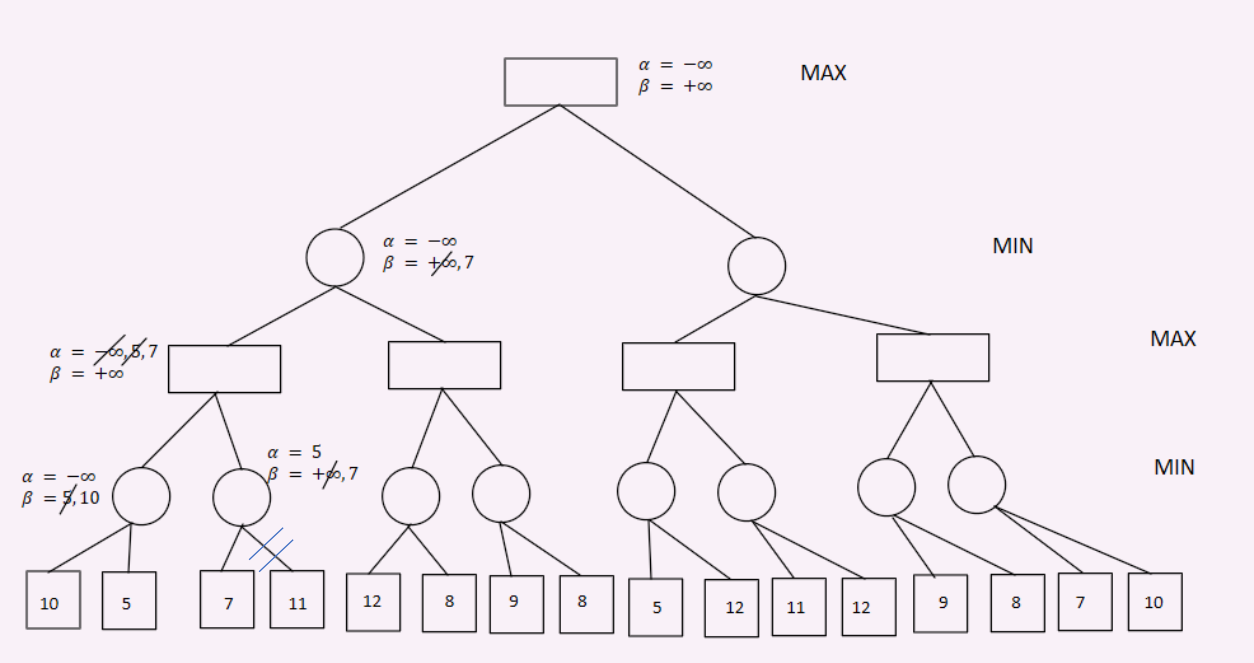


Bajamos al siguiente nodo hijo, cuando subimos un valor debemos saber el tipo de nodo al que vamos a subir el mismo, al contrario que cuando bajamos un valor donde no se tiene en cuenta el tipo de nodo, simplemente lo bajamos.

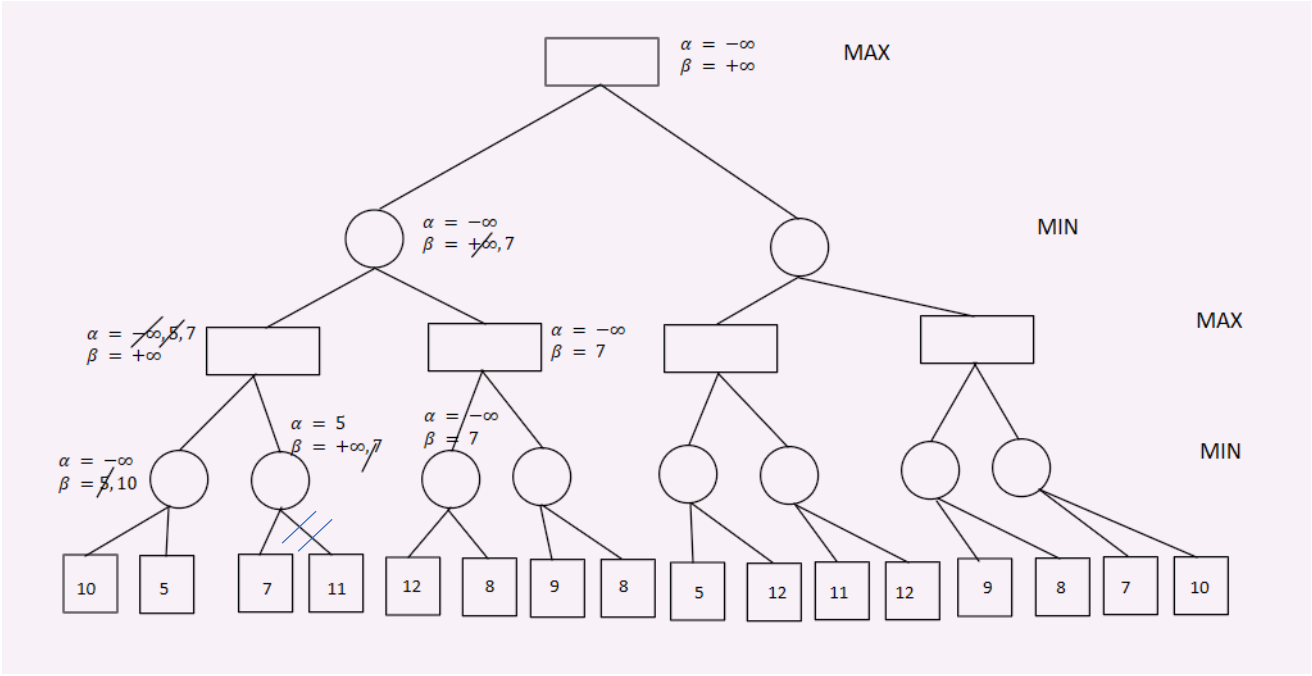


Al igual que antes nos encontramos en un nodo min por lo que modificaremos el valor de la β. Aplicamos la siguiente formula $\beta = min(7, +\infty) = 7$. Cuando estemos modificando tanto al alpha como la beta de un nodo debemos tener en cuenta lo siguiente: "Si en un nodo no se cumple que α ≥ β entonces no comprobamos el resto de nodos hijos". En este caso sucede que β = 7 y α = 5 por lo que no debemos calcular el nodo 11.

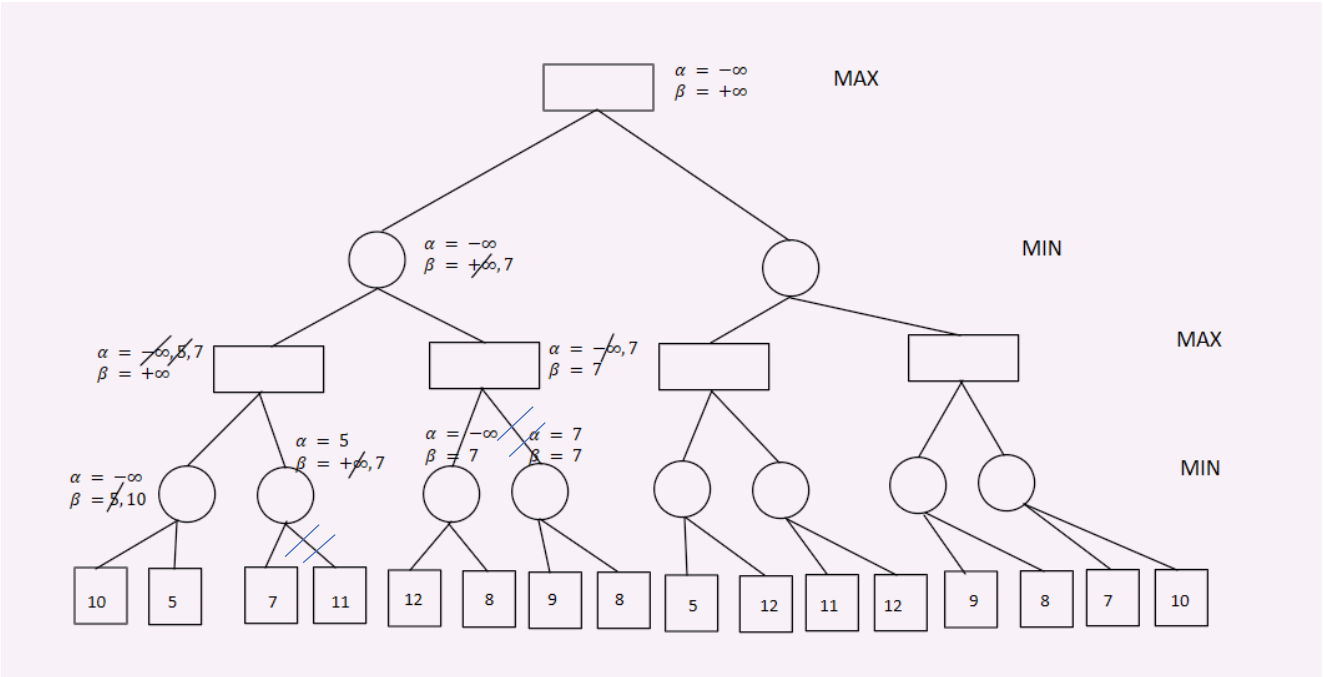
Subimos el valor del nodo al nodo padre, queremos subirlo a un nodo max por lo que debemos modificar la α, aplicamos la siguiente formula $\alpha = \max(5, 7) = 7$. Procedemos igual para subirlo al siguiente nodo padre que en este caso en un nodo min, aplicamos la siguiente formula $\beta = \min(7, +\infty) = 7$.



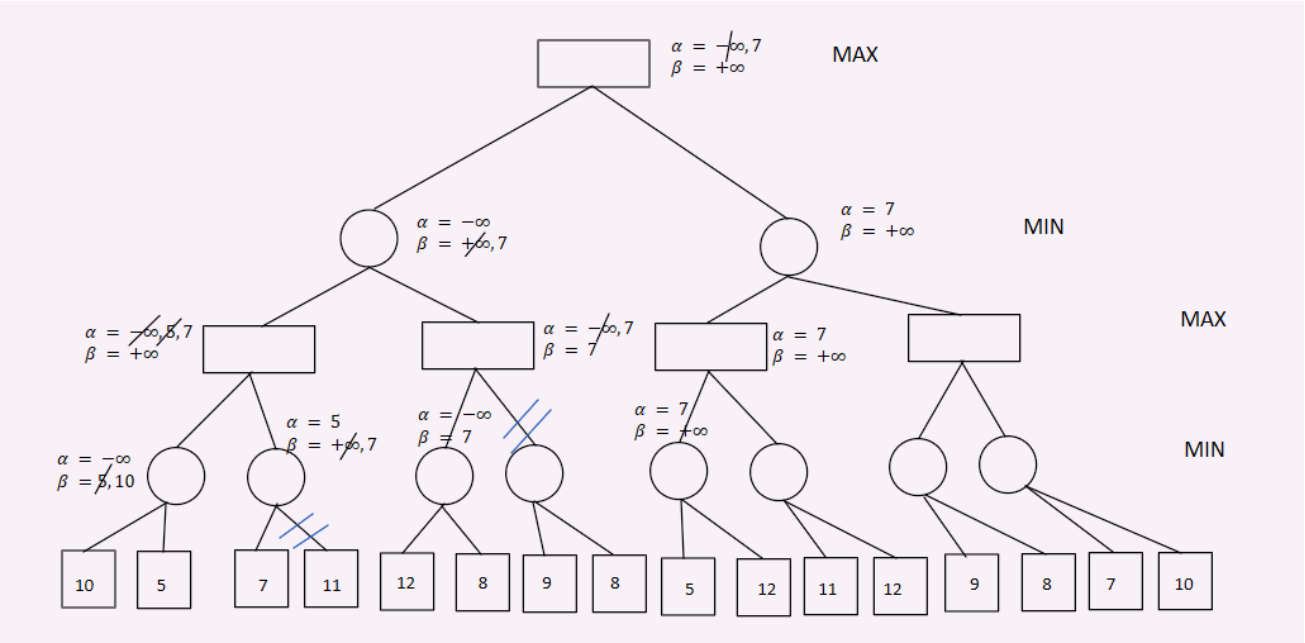
Bajamos al siguiente nodo hoja y procedemos igual que antes.



Ahora estamos en un nodo min, por lo que debemos modificar el valor de β, aplicamos la siguiente formula $\beta = \min(12, 7) = 7$. Vamos al siguiente nodo hijo y procedemos igual que antes $\beta = \min(8, 7) = 7$. En este caso no se modifica el valor del nodo, lo subimos al nodo padre maximizador, debemos modificar el valor de α, por lo aplican os la siguiente formula $\alpha = \max(7, -\infty) = 7$.

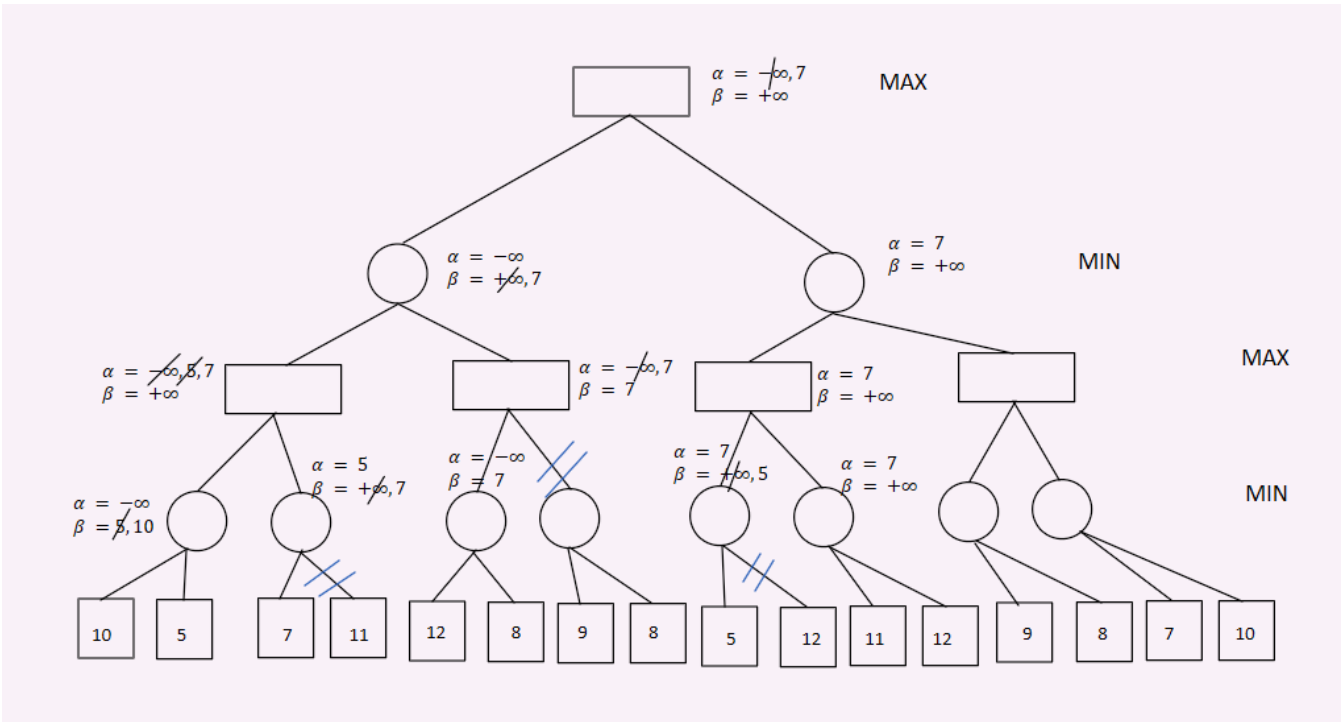


Como α ≥ β no comprobamos las siguientes ramas. Subimos el valor del nodo hasta el nodo padre y continuamos el algoritmo. Aplicamos la siguiente fórmula $\alpha = \max(7, -\infty) = 7$.



Estamos en un nodo min por lo que debemos modificar el valor de β, aplicamos la siguiente formula $\beta = \min(5, +\infty) = 5$. Ahora α = 7 y β = 5, es decir, α ≥ β por lo que no comprobamos el resto de ramas del árbol.

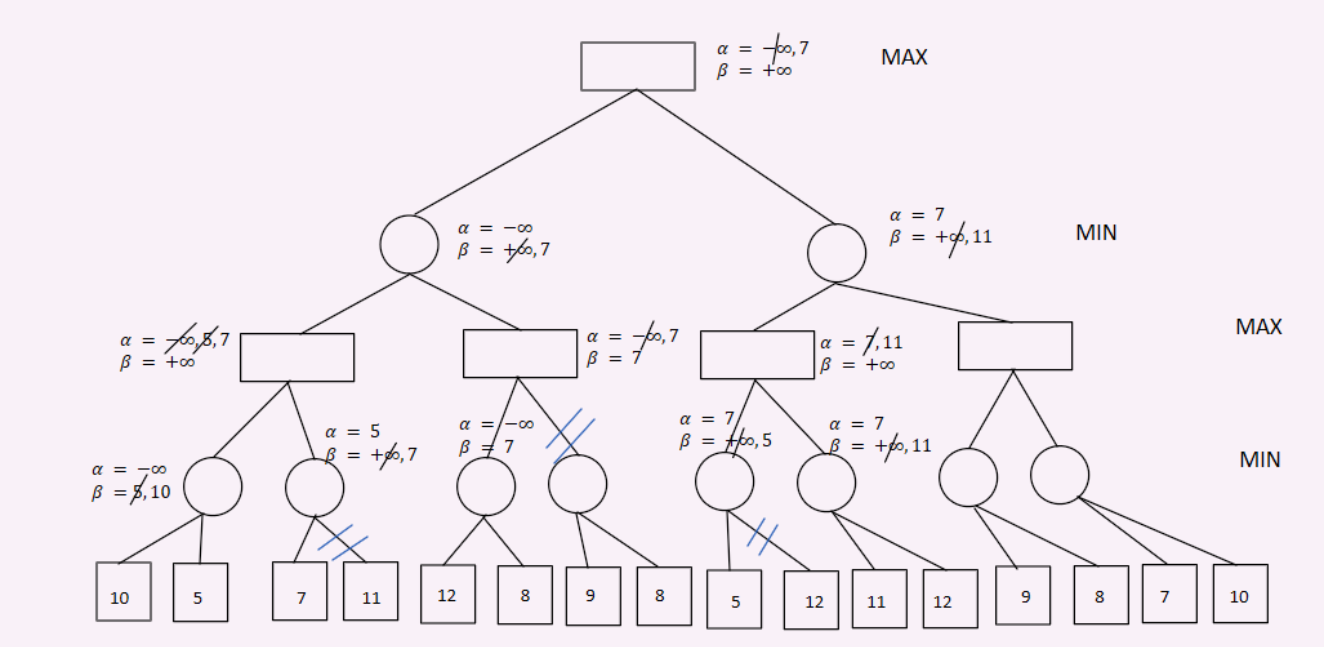
Lo subimos al nodo padre, en este caso es un nodo maximizador por lo que debemos modificar el valor de α, aplicamos la siguiente formula $\alpha = \max(7, 5) = 7$.



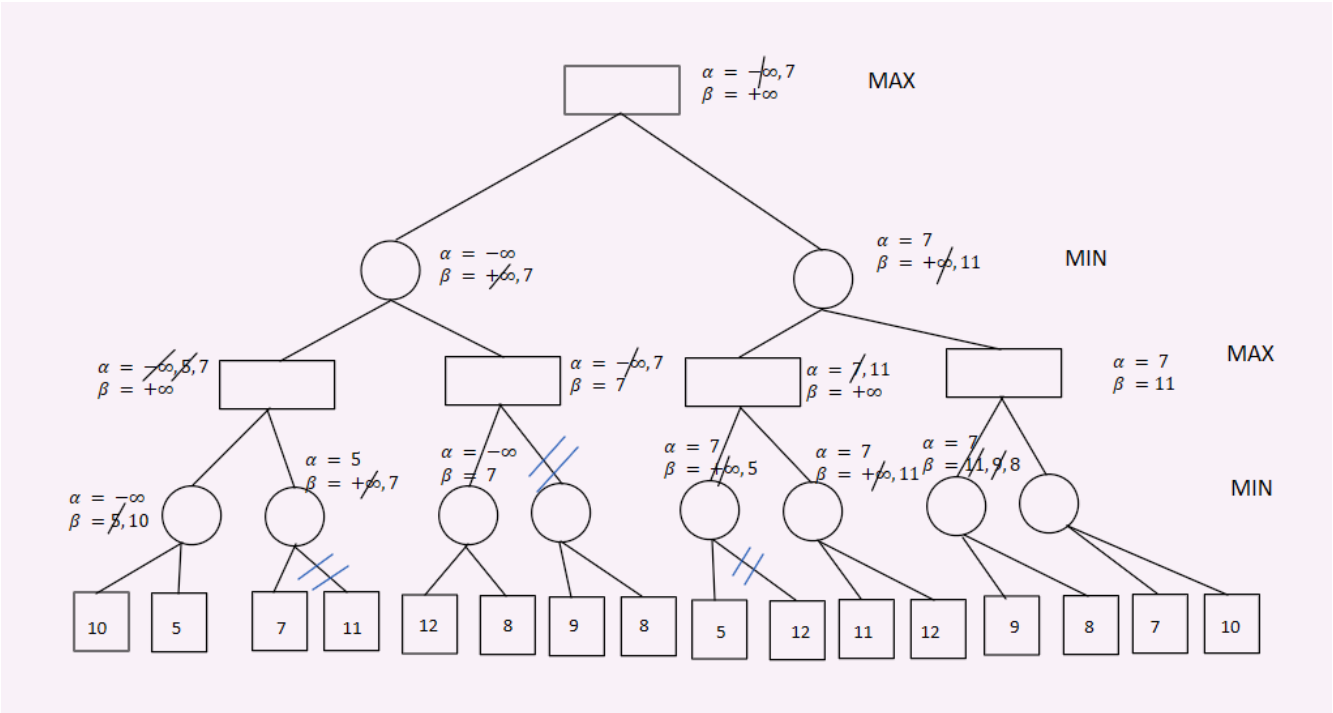
Estamos en un nodo min por lo que queremos modificar el valor de β, aplicamos la siguiente formula $\beta = \min(11, +\infty) = 11$. Aplicamos el mismo procedimiento para el siguiente nodo hijo $\beta = \min(12, 11) = 11$.

Subimos el valor del nodo al nodo padre que en este caso es un nodo max, calculamos el valor de α, aplicamos la siguiente formula $\alpha = \max(11, 7) = 11$.

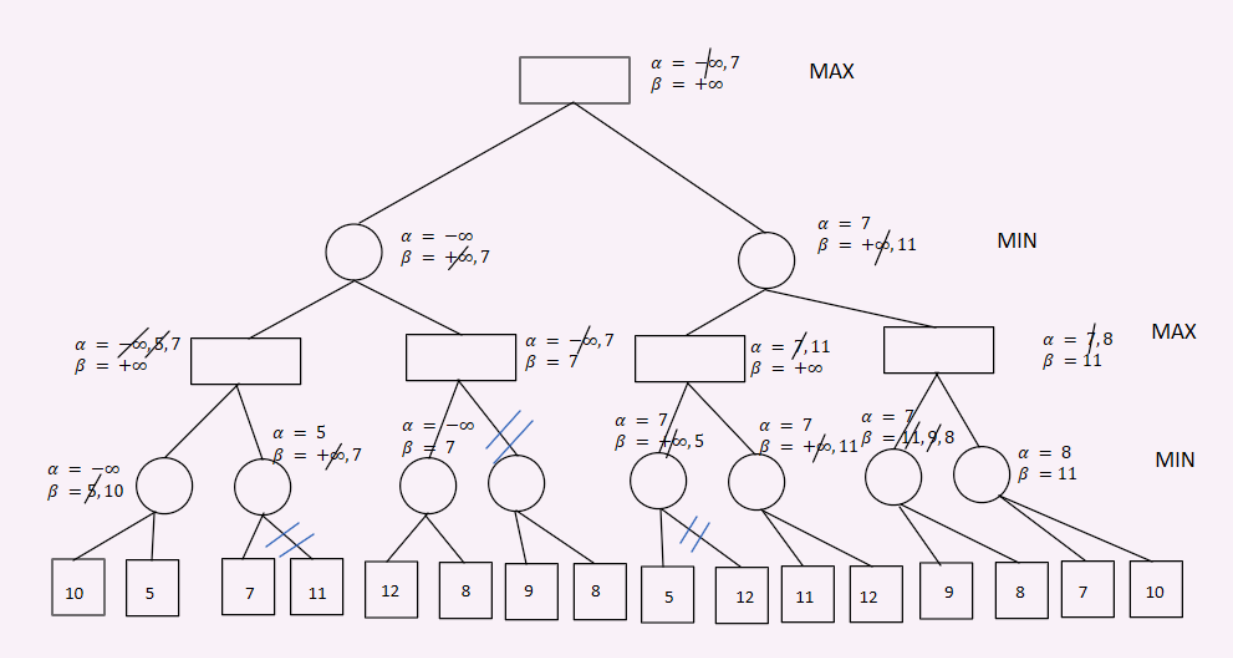
El siguiente nodo padre es un nodo min, queremos calcular el valor de β, aplicamos la siguiente formula, $\beta = \min(11, +\infty) = 11$.



Bajamos el valor del nodo padre hasta el nodo disponible más a la izquierda. Estamos en un nodo min, por lo que queremos modificar el valor de β, aplicamos la siguiente formula $\beta = \min(9, 11) = 9$. Aplicamos el mismo procedimiento para el siguiente nodo hijo $\beta = \min(8, 9) = 8$.

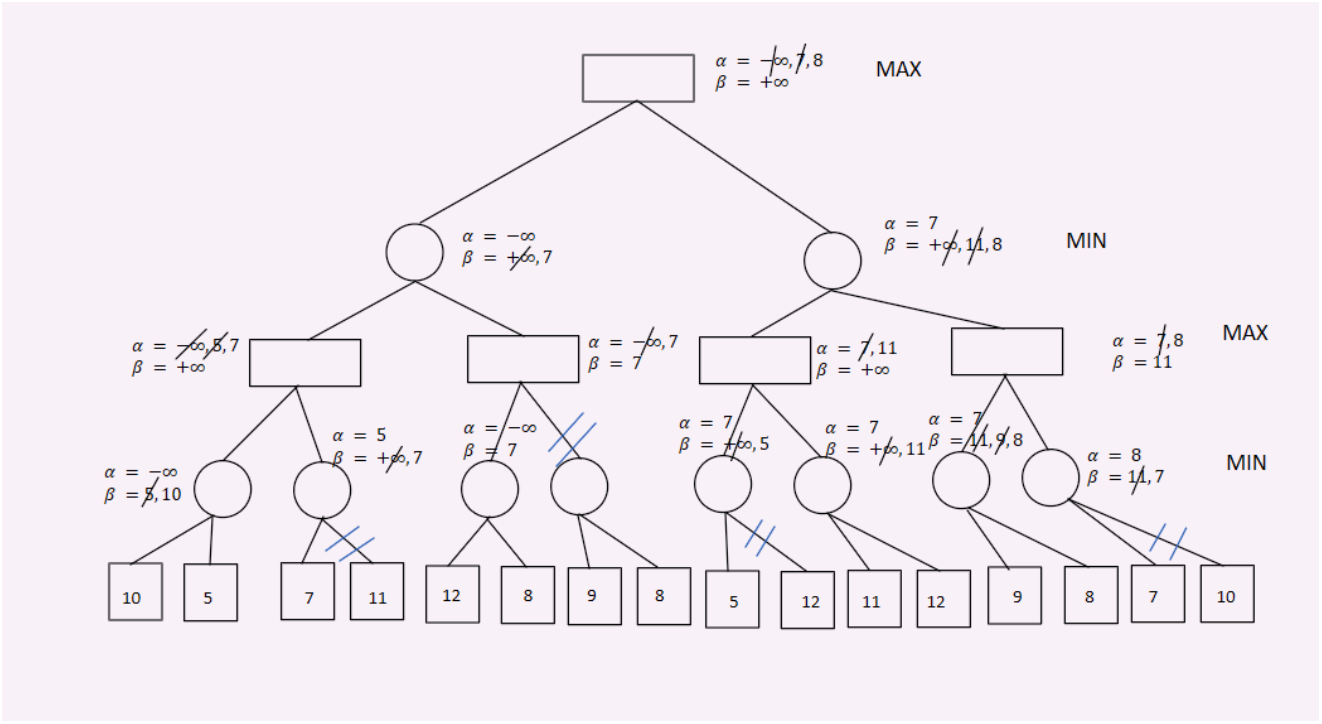


Ahora estamos en un nodo max por lo que debemos modificar el valor de α, aplicamos la siguiente formula $\alpha = \max(8, 7) = 8$.



Estamos en un nodo min por lo queremos modificar el valor de β, aplicamos la siguiente formula $\beta = \min(7, 11) = 7$. Como α ≥ β no comprobamos el siguiente nodo hijo.

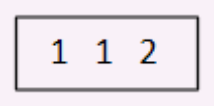
Ahora subimos el valor del nodo al nodo padre. Dicho nodo padre es un nodo max por lo que queremos modificar el valor de α, aplicamos la siguiente formula $\alpha = \max(7, 8) = 8$. Continuamos subiendo el valor, en este caso estamos en un nodo min aplicamos la siguiente formula $\beta = \min(8, 11) = 8$. Subimos al siguiente nodo padre, es un nodo max por lo que aplicamos la siguiente formula, $\alpha = \max(7, 8) = 8$.



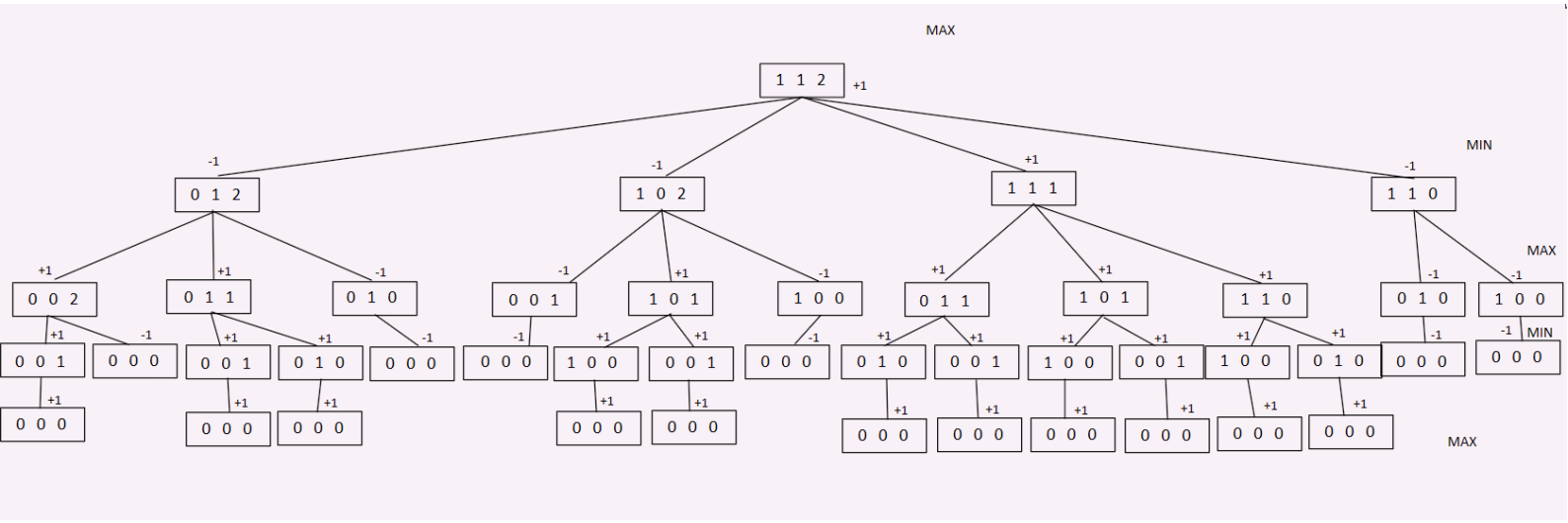
2.- Problema de las monedas: En una mesa hay k montones de monedas, con n monedas cada uno. Dos jugadores en turnos alternos pueden jugar retirando de uno de los montones una o más de las restantes. Pierde el jugador que retire la última moneda. (a) Formular el problema del juego dando una representación formal adecuada de los estados y jugadas posibles del mismo; (b) En el juego con k = 3, n1 = n2 = 1, n3 = 2, aplique el metodo MinMax para determinar las estrategias optimas; (c) Ídem con el metodo de Poda alpha-beta, indicando en el árbol de la parte (b) que ramas han sido podadas.

En el problema cada uno de los estados serán las posibles situaciones del juego en la que nos encontremos, además las operaciones de cambio de estados serán aquellas en las que los jugadores retiren monedas de los montones. Las operaciones tienes restricciones en el momento en el que los jugadores solo pueden retirar en una misma jugada monedas de un único montón.

Por ejemplo si disponemos del siguiente ejemplo en el que tenemos tres montones k = 3, n1 = n2 = 1, n3 = 2, nuestro estado inicial sería el siguiente:



A continuación voy a dibujar el árbol del problema para poder resolverlo mediante MinMax.

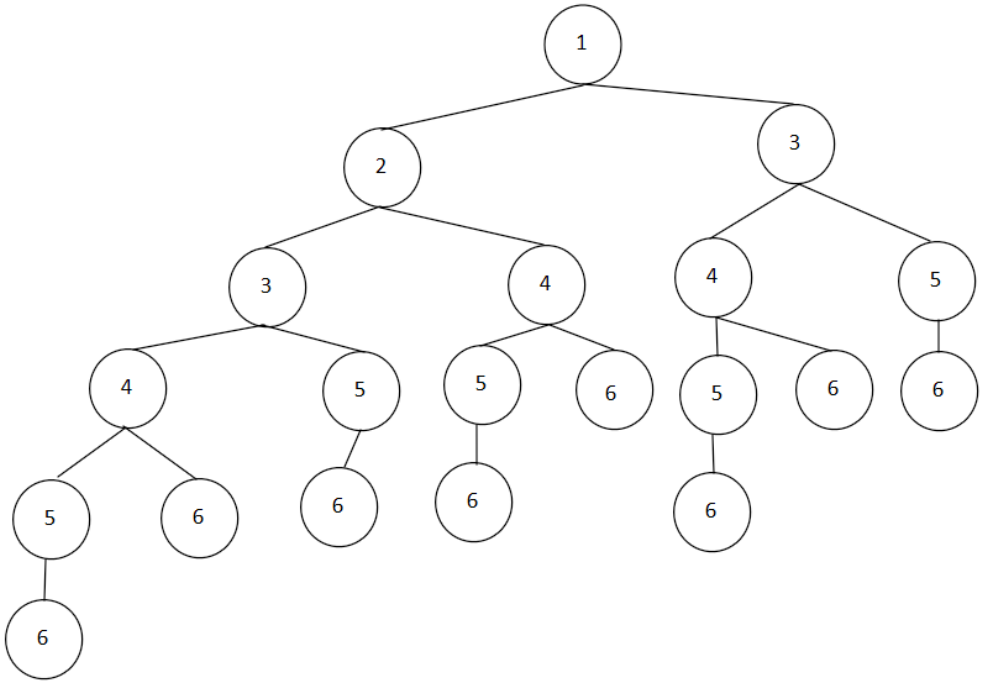


Marco las puntuaciones en las hojas dependiendo de quien haya ganado(jugador1 = +1, jugador2 = -1). Una vez tengo esas puntuaciones vamos subiendo con el algoritmo MinMax, es decir, cuando le toque al jugador1(Max) miraremos sus hijos y cogeremos la máxima puntuación mientras que si el jugador2(Min) miraremos sus hijos y cogeremos la mínima puntuación.

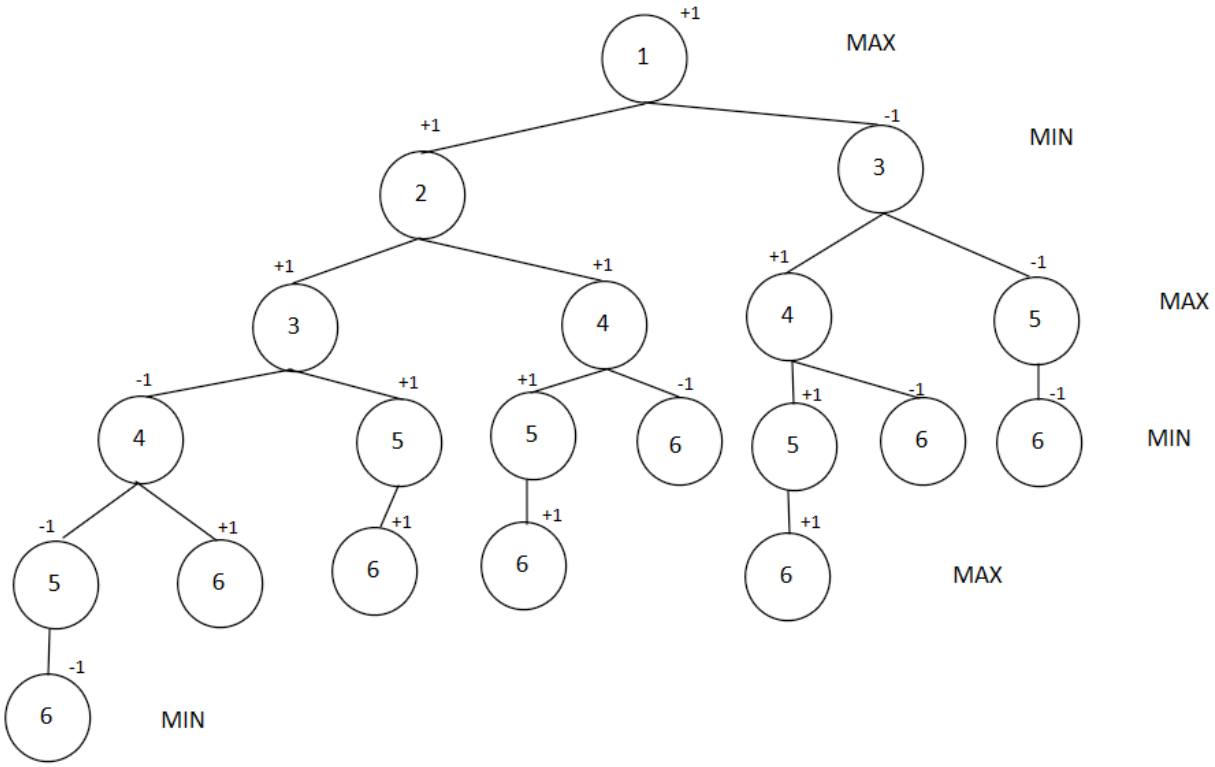
En este caso la mejor estrategia para que gane el jugador 1 seria 1-1-2, 1-1-1, …, 0-0-0. En el momento en el que el jugador 1 elimine una moneda del montón que contiene dos, automáticamente se ha asegurado la victoria pues el jugador 2 solo podría eliminar fichas de uno en uno hasta perder.

3.- Problema de Nim: Cada uno de los jugadores debe ir contando desde seis hasta cero alternadamente. En su turno, cada jugador puede seguir la cuenta de uno en uno o de dos en dos. Determinar la estrategia optima.

En cuanto al problema se representará mediante un grafo el cual a continuación se resolverá mediante el algoritmo MinMax.



Determino que nodos corresponderán al jugador maximizador y que nodos corresponderán al nodo minimizador y comienzo el algoritmo.

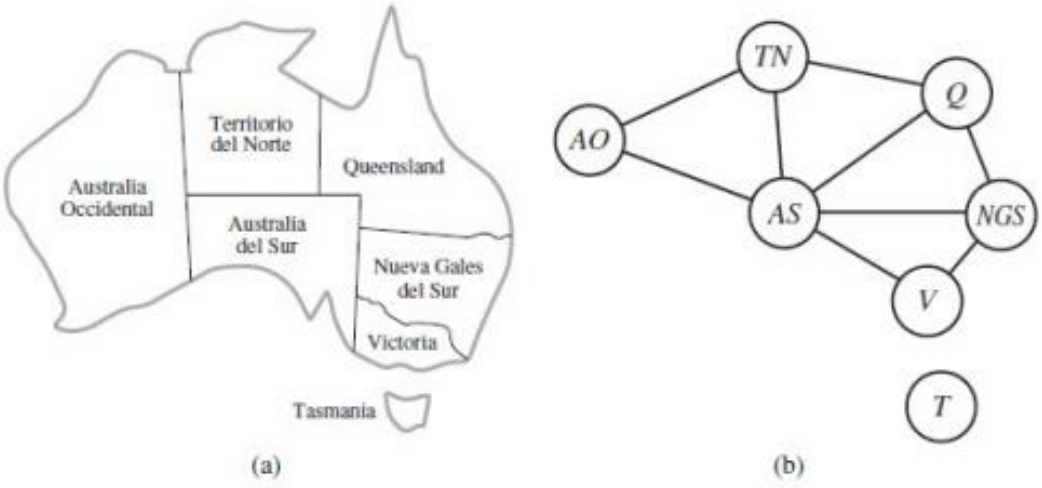


PROBLEMAS DE SATISFACCION DE RESTRICCIONES

Los problemas de satisfacción de restricciones(PSR) vienen definidos por un conjunto de variables: X1, X2, X3, …, Xn, así como un conjunto de restricciones C1, C2, C3, …, Cn. A su vez cada variable Xi tiene asignada un dominio no vacío de valores posibles. El objetivo principal de este tipo de problemas es asignar valores aceptables al conjunto de variables de tal modo que cumplan con las restricciones impuestas. De esta manera, aquellas asignaciones que no violan ninguna restricción se les denomina asignacion de consistente o legal. Se dice que aquella asignacion es completa si cada variable satisface todas las restricciones.

Por ejemplo, el mapa de Australia dispone de un total de siete regiones, tal y como se muestra en el siguiente mapa:

El objetivo consiste en colorear, empleando solo tres colores, cada una de las regiones de forma que aquellas regiones vecinas no tengan el mismo color. La mejor forma de ver este tipo de problemas es mediante un grafo de restricciones.



El objetivo es transformar un problema de satisfacción de restricciones en un problema de búsqueda. ¿Pero cómo lo podemos hacer? El primer paso es definir las componentes que definimos en el primer tema, pero en este caso para un PSR. Para verlo más claro vamos a definir esas componentes teniendo en cuenta el ejemplo del mapa de Australia:

1. **Estado inicial:** inicialmente no hay ninguna región del mapa pintada, por lo que la asignacion es vacía {}.
2. **Operaciones de cambio de estado:** un valor se puede asignar a cualquier variable no asignada siempre y cuando no suponga un conflicto con las restricciones impuestas. En el caso del mapa de Australia, si elegimos como primer nodo Australia del Sur y la pintamos de rojo, como sucesor podríamos elegir Territorio del Norte con la condicion de no pintarlo de rojo pues violaría la restricción impuesta.
3. **Estado meta:** en este caso se podría considerar como estado meta rellenar el mapa cumpliendo con todas las restricciones.
4. **Coste:** para este ejemplo el coste sería constante para cada paso.

TIPOS DE PROBLEMAS DE ASIGNACION DE RESTRICCIONES

Según el tipo de **dominio**:

- **Variables discretas:** pueden ser valores finitos o infinitos. Valores finitos como por ejemplo el problema anterior, en el que solo podíamos usar tres colores diferentes; e infinitos por ejemplo la asignacion de tareas de un proyecto que puede durar mucho tiempo.

Según el número de **variables**:

- **Restricciones unarias:** son restricciones que afectan solo a una variables, por ejemplo que Nueva Gales del Sur deba tener el color rojo.
- **Restricciones binarias:** relaciona dos variables. Como el caso anterior, dos regiones vecinas no pueden tener el mismo color.
- **Restricciones n-arias:** más de dos variables implicadas.

Según el tipo de **restricción**:

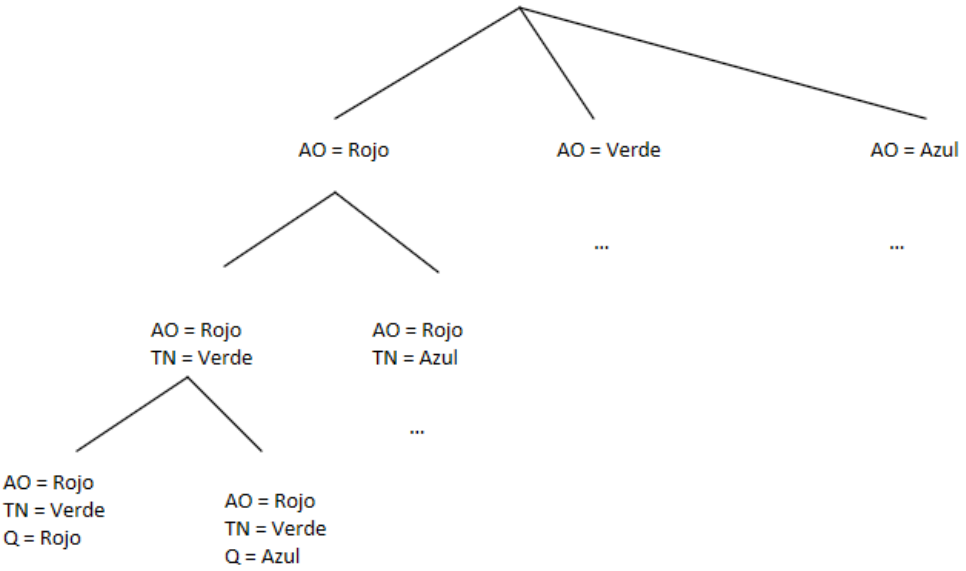
- **De obligatorio cumplimiento:** obligatoriamente, aquellas regiones adyacentes no pueden representar el mismo color.
- **De preferencia:** indica que soluciones son preferidas.

Como henos dicho antes los problemas de satisfacción de restricciones pueden reformularse como problemas de búsqueda en espacio de estados, con diversas particularidades favorables y con heurísticas naturales: metodo constructivo o incremental(backtracking) o metodo de reparación heurística.

METODO CONSTRUCTIVO O INCREMENTAL

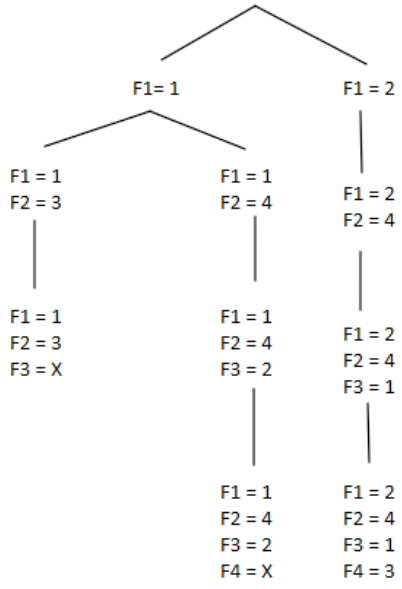
Una propiedad crucial en todos los PSR es la conmutatividad. Un problema es conmutativo si el orden a de aplicación de cualquier conjunto de acciones no hace ningún efecto sobre el resultado. El termino de búsqueda con vuelta atrás se utiliza para la búsqueda primero en profundidad que elige valores a la vez para una variable y vuelve para atrás cuando una variable no tiene ningún valor legal para asignarle. Mas adelante veremos que a estos métodos podemos añadirle heurísticas naturales que mejoren la eficiencia.

Veamos este metodo con el problema de colorear Australia. Tenemos siete variables y cada una puede tomar tres valores, es decir, podemos tener un total de 21 asignaciones.



El orden en el que realicemos la asignacion no afecta a la meta. Cada vez que hacemos una asignacion estamos añadiendo una nueva variable por lo que nunca puede haber bucles y tampoco puede haber una rama infinita. Cada rama tiene como máximo el número de variables.

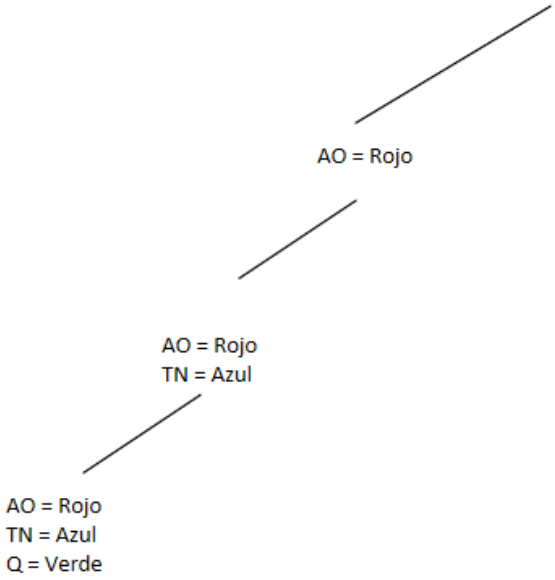
Otro ejemplo puede ser el problema de las cuatro reinas, donde tenemos como variable las filas y en el dominio las columnas.



En la resolución incremental de PSR, se pueden considerar heurísticas naturales genéricas basadas solo en la estructura de los problemas que mejoran su eficiencia:

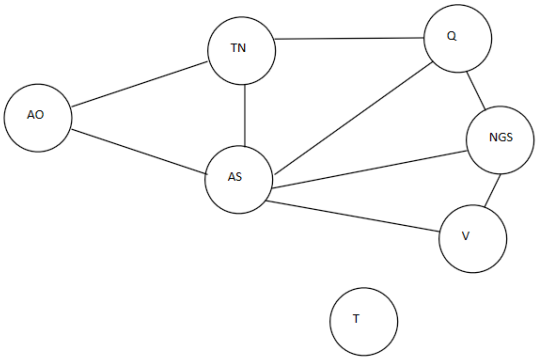
1. Orden de asignacion a las variables.
2. Orden de valores a asignar en cada dominio.
3. Propagación de información mediante las restricciones.
4. Orden adecuado en la vuelta atrás, cuando haya que hacerla.
5. Aprovechamiento de la estructura del problema.

Vamos a comentar ahora el problema de realizar una asignacion de variables en un orden ineficiente. Imaginemos el siguiente orden para realizar la asignacion: AO, TN, Q, NGS, V, T, AS. Si nos fijamos hemos dejado AS para el final, lo que provocara un problema ya que es la región más conectada.



Al seleccionar un orden a priori malo, lo que provocamos es que aumente la complejidad lo que implica más gasto en memoria.

Para resolver problemas PSR lo más eficiente posible vamos a aplicar heurísticas naturales. Además para ver mejor las restricciones emplearemos lo que se denomina como grafo de restricciones.



Disponemos de dos heurísticas naturales:

- **MVR(menos valores restantes):** consiste en escoger la variable con un menor número de valores legales posibles. Dicho de otro modo, escoger la variable que elimine el menor número de valores posibles para las variables contiguas. Por ejemplo, supongamos que, partiendo con el mapa de Australia, tanto la región Australia Occidental como Tasmania están coloreadas de rojo y verde, respectivamente. Las regiones adyacentes son Australia del Sur y Queensland. Por ello, ¿cuál debemos colorear antes? Según MVR debemos elegir aquel nodo que menor número de posibles valores presente. En el caso de Australia del Sur, como el resto de nodos adyacentes están pintados a rojo y verde, el color restante es el azul, mientras que Queensland puede colorearse con rojo o azul, luego elegiremos como siguiente nodo Australia del Sur, pues es aquel nodo que menos valores posibles presenta.
- **Grado de conexión o grado heurístico(GC o GH):** Intenta reducir el factor de ramificación sobre futuras opciones, seleccionando la variable, entre las variables no asignadas, que este implicada en el mayor número de restricciones o lo que es lo mismo que tenga más conexiones en el grafo de restricciones.

Si nos fijamos en estas dos heurísticas se basan únicamente en la elección de las variables? Que pasa entonces con el valor de los mismos?. La heurística VMN(Valor Menos Restringido) da preferencia a aquel valor que excluya las pocas opciones de las variables en el grafo de restricciones. Por ejemplo AO = Roja y TN = Verde. Nuestra siguiente opción es para Q = Azul, esta sería una mala opcional que elimina el ultimo valor legal de3 SA que es vecino de Q.

Volviendo al problema de colorear Australia los grados heurísticos o de conexión serían los siguientes:

$g(t) = 0$

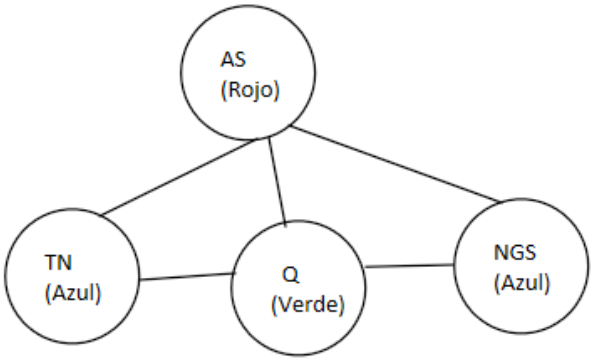
$g(AO) = g(V) = 2$

$g(TN) = g(NGS) = g(Q) = 3$

$g(AS) = 5$

Esto nos indica que la mejor opción es empezar por AS ya que tiene un mayor grado de conexión.

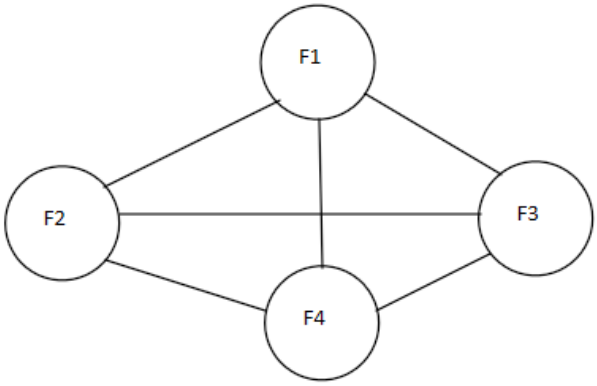
Si nos fijamos a Q le podemos dar solo el valor de Verde mientras que a NGS el Verde y el Azul.



Según la heurística que hemos visto, MVR de Q es 1 y el MVR de NGS seria 2, pero además si nos fijamos en VMN si damos el color Verde a NGS, Q no tendría ningún valor legal disponible. Por lo tanto debemos asignar a Q el Verde y el Azul a NGS.

Vamos a ver el ejemplo de las 4-reinas. Para ello al igual que antes disponemos de cuatro variables F1, F2, F3, F4; como dominio las posibles columnas donde colocar dichas reinas y como restricción, las reinas no pueden estar en la misma fila, columna y diagonal.

En primer lugar construimos el grafo de restricciones:



A la hora de elegir el primer valor, estamos empatados en la heurística MVR y en la heurística GC. Por lo que si aplicamos VMN y seleccionamos F1 = 1 ocurre lo siguiente:

F1 = 1

F2 = (~~1, 2~~, 3, 4)

F3 = (~~1~~, 2, ~~3~~, 4)

F4 = (~~1~~, 2, 3, ~~4~~)

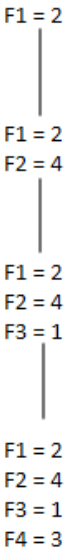
Pero si seleccionamos F1 = 2:

F2 = (~~1, 2, 3~~, 4)

F3 = (1,~~ 2~~, 3, ~~4~~)

F4 = (1,~~ 2,~~ 3, ~~4~~)

Como vemos con F1 = 1 quitamos dos posibilidades a todas, pero si cogemos F1 = 2 quitamos solo una posibilidad a F4 por lo que la primera asignacion será F1 = 2. Asignacion completa si realizar backtracking.



METODO DE ASIGNACION HEURÍSTICA

En este metodo no se sigue una táctica de asignacion incremental, sino una táctica de asignacion completa, que se va corrigiendo progresivamente. El metodo consiste en empezar con una asignacion completa arbitraria, que constituiría el nodo inicial. Examinar si es solución, en cuyo caso presentarla y terminar. Si no lo es, elegir una variable entre las que más restricciones cumplan, y generar sucesores cambiando el valor de dicha variable por otro. Repetir el proceso hasta que no haya conflictos o se llegue a un límite prestablecido del número de iteraciones, en cuyo caso se presentaría fallo. Veamos este metodo con el problema de las cuatro reinas.

En primer lugar establecemos una asignacion arbitraria como por ejemplo loa siguiente:

F1 = 1

F2 = 1

F3 = 1

F4 = 1

Como podemos observar la asignacion arbitraria no es solución, existen tres conflictos, es decir, existen tres restricciones incumplidas por lo que debemos generar los estados sucesores.

F1 = 2	F1 = 3	F1 = 4
F2 = 1	F2 = 1	F2 = 1
F3 = 1	F3 = 1	F3 = 1
F4 = 1	F4 = 1	F4 = 1

Como Podemos observar, todos presentan un conflicto por lo tanto da igual que rama desarrollar. En este caso desarrollamos la primera y generamos sus sucesores:

F1 = 2	F1 = 2	F1 = 2
F2 = 2	F2 = 3	F2 = 4
F3 = 1	F3 = 1	F3 = 1
F4 = 1	F4 = 1	F4 = 1

Ahora el ultimo estado no presenta conflictos por lo que lo desarrollamos.

F1 = 2	F1 = 2	F1 = 2
F2 = 4	F2 = 4	F2 = 4
F3 = 2	F3 = 3	F3 = 4
F4 = 1	F4 = 1	F4 = 1

El Segundo estado no presenta conflictos por lo que desarrollamos sus sucesores.

F1 = 2	F1 = 2	F1 = 2
F2 = 4	F2 = 4	F2 = 4
F3 = 3	F3 = 3	F3 = 3
F4 = 2	F4 = 3	F4 = 4

Como podemos observar no hemos llegado a la solución. Veamos una comparación entre el metodo incremental y el metodo de reparación heurística.

En primer lugar el metodo incremental es completo, a diferencia de la reparación heurística que puede estar dando vueltas continuamente. Además el metodo de reparación es mucho menos costoso computacionalmente permitiendo resolver problemas de mayor tamaño a cambio de una menor seguridad.

Por otro lado el metodo de reparación heurística se permite cambiar las restricciones dinámicamente, en el curso de la ejecución lo que es muy impredecible en problema de planificación en tiempo y espacio. Por último, ambos métodos carecen de ser óptimos.

Para acabar el tema veamos el concepto de arco de conexión o arco consistente. Permite mejorar la eficiencia de los métodos constructivos. Cuando hablamos de arcos nos referimos a un arco dirigido en el grafo de restricciones. Como por ejemplo el arco de AS a NGS, en el problema de colorear Australia. Considerando los dominios actuales de AS y NGS, el arco será consistente si para todo valor de x de AS, hay algún valor de NGS que es consistente con x.

Veamos un ejemplo. Tenemos un robot que hace n tareas que se efectúan en unidades de tiempo:

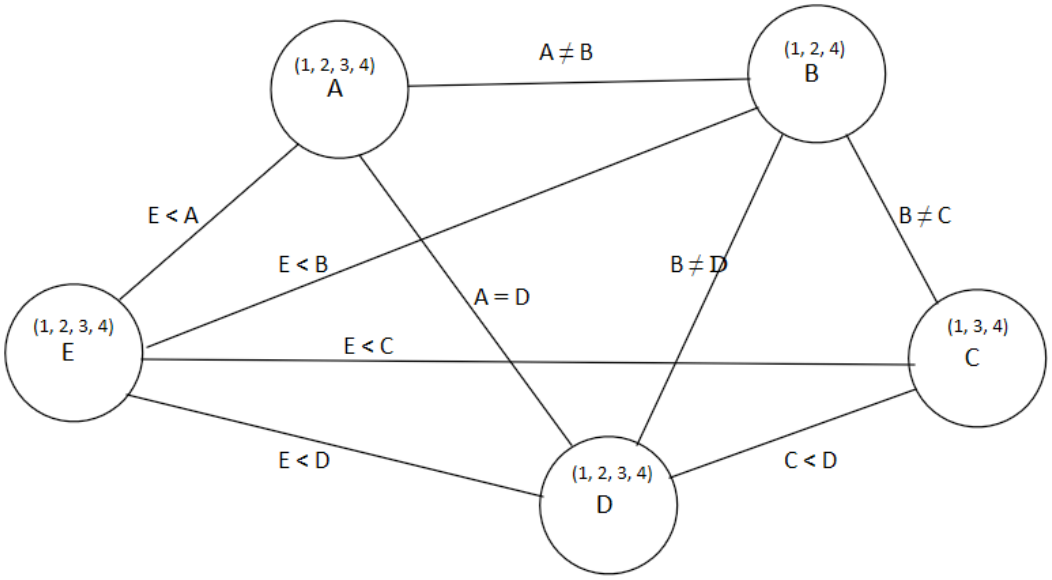
Variables(tareas): {A, B, C, D, E}.

Dominio(tiempo): {1, 2, 3, 4}.

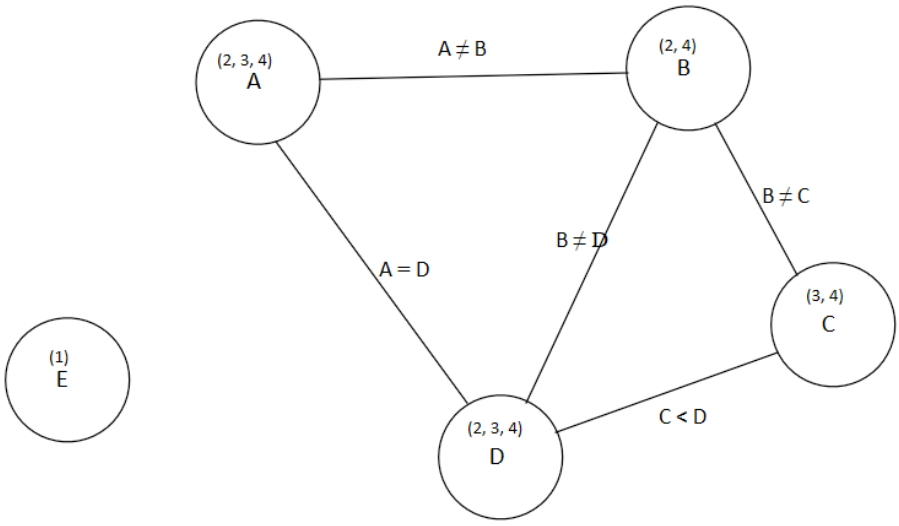
Restricciones:

- B no puede ser 3
- C no puede ser 2
- A y B no pueden ser simultaneas
- B y C no pueden ser simultaneas
- B y D no pueden ser simultaneas
- A y D han de ser simultaneas.
- C debe ir antes que D
- E debe ir la primera

Dibujamos el grafo de restricciones:



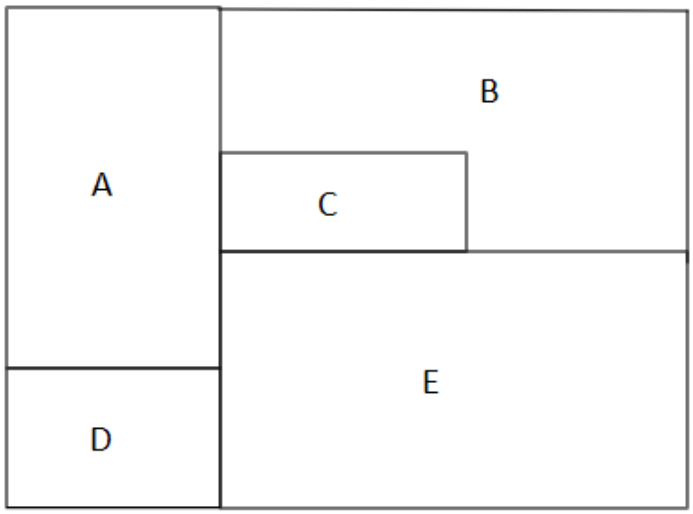
Imaginemos que asignamos el valor 1 a E, entonces reducimos el problema en un subproblema relativo a A, B, C, D.



En este caso un arco de conexión es cualquier punto que me lleve a otro. Por ejemplo, E y el resto. Siempre y cuando podamos dividir el problema en varios arcos permite reducir el factor de ramificación ya que trata os arboles más pequeños.

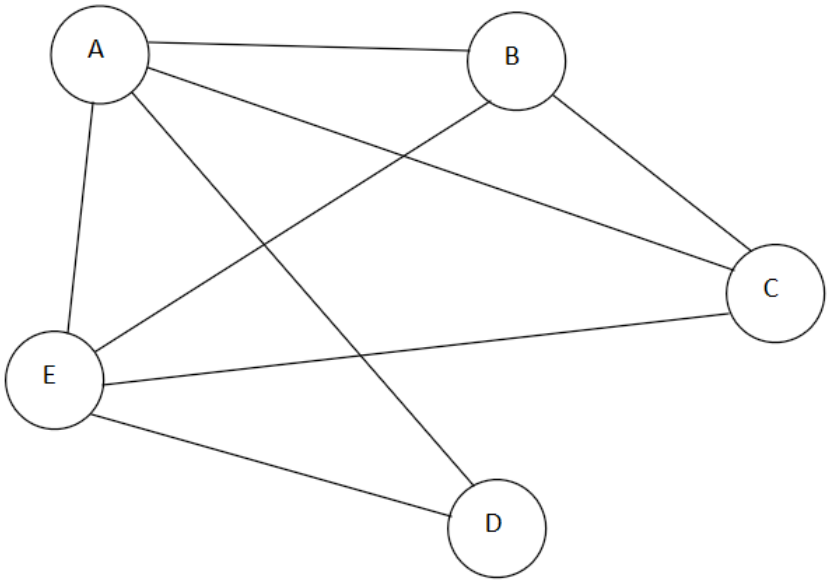
EJERCICIOS

1.- Problema de coloración: Dado el mapa de la figura (a) Plantee formalmente como PSR el problema de su coloreado; (b) dibuje el gafo de restricciones correspondiente, etiquetando sus nodos y arcos; (c) aplique el procedimiento AC-3 para podar los dominios; (d) resuelva el problema constructivamente.



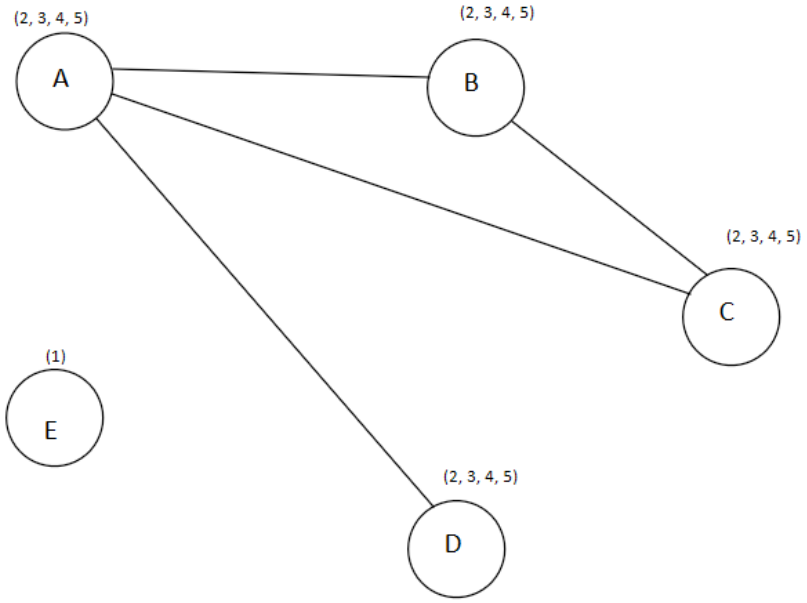
El objetivo es colorear el mapa mostrado de forma que las regiones adyacentes no tengan el mismo color. Por otro lado cuantos menos colores empleemos en dicha coloración mejor será la solución. En cuanto a los estados, sabemos que inicialmente no disponemos de ninguna región del mapa coloreada por lo que el estado inicial estará vacío {}, por otro lado el estado final será el mapa coloreado habiendo respetado las restricciones impuestas. Las operaciones de cambio de estado será la coloración de uno de los territorios de forma que si empleamos un color para colorear una zona, no podremos usar el mismo colora para otra región adyacente.

En cuanto al grafo de restricciones, se presenta a continuación. Los arcos entre nodos representan la adyacencia de las regiones en el mapa, así dos regiones adyacentes serán representadas mediante dos nodos unidos por un arco.

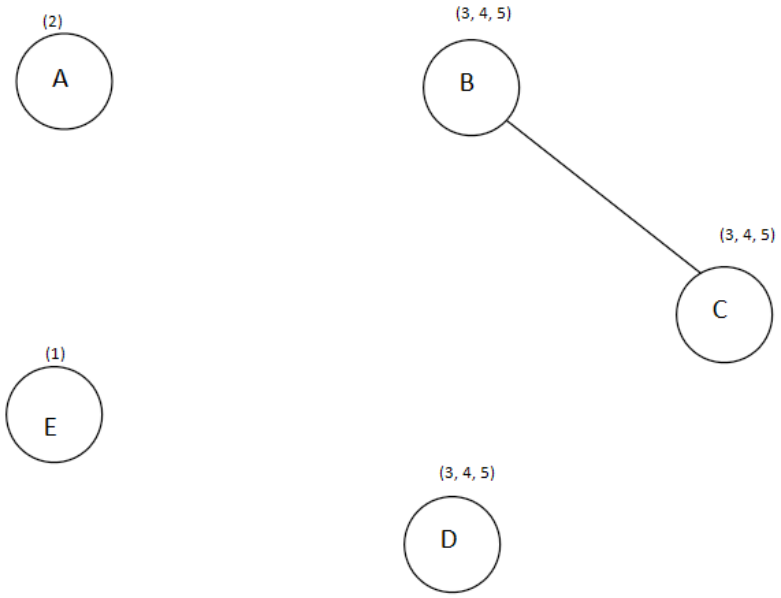


Inicialmente disponemos de el siguiente domino {1,2 3, 4, 5}, imaginando que dicho dominio son los colores con los que podemos colorear el mapa. De esta forma si asignamos dicho dominio a nuestro grafo, podamos el grafo mediante el algoritmo de consistencia de arco.

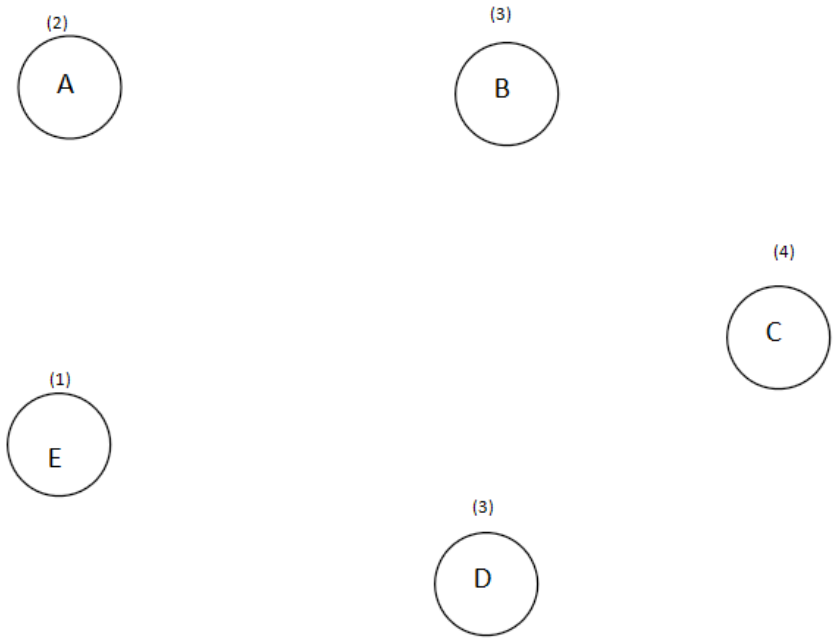
Imaginemos que asignamos el valor 1 al nodo E, entonces estaríamos reduciendo el problema a un subproblema de menor complejidad:



Una vez hemos asignado el color 1 al nodo E, reducimos el factor de ramificación del problema. Ahora podemos asignar un nuevo color a algún nodo. Vamos a asignar el color 2 al nodo A, con eso al igual que antes reducimos el factor de ramificación.



Ahora podemos asignar un mismo color al nodo D y al nodo B o nodo C. Aplicamos el color 3 a ambos y seguidamente aplicamos el color 4 al último nodo.



2.-Cuestion de examen: Problemas de satisfacción de restricciones: en qué consisten, cuál es su relación con los problemas de búsqueda, que características particulares tienen, con que métodos pueden resolverse.

Los problemas de satisfacción de restricciones(PSR) vienen definidos por un conjunto de variables: X1, X2, X3, …, Xn, así como un conjunto de restricciones C1, C2, C3, …, Cn. El objetivo principal de este tipo de problemas es asignar valores aceptables al conjunto de variables de tal modo que cumplan con las restricciones impuestas. De esta manera, aquellas asignaciones que no violan ninguna restricción se les denomina asignacion de consistente o legal. Se dice que aquella asignacion es completa si cada variable satisface todas las restricciones. El objetivo

es transformar un problema de satisfacción de restricciones en un problema de búsqueda, para ello debemos definir las misma componentes que cuando definíamos un problema de búsqueda tanto informada como desinformada. Estos problemas tienen la particularidad de que presentan restricciones o preferencias sobre unos nodos que sobre otros.

En cuanto a la resolución de este tipo de problemas, disponemos de dos métodos, un método constructivo y un metodo de asignacion heurística. En el método constructivo se sigue una asignacion iterativa del dominio es decir, se van asignado valores al dominio a medida que avanzamos en el problema. Por el contrario en el método de asignacion heurística se asigna por completo y de forma arbitraria los valores a todas las variables del problema.