
# Solución de problemas con búsqueda

Sistemas Inteligentes

Dr. Víctor de la Cueva
vcueva@itesm.mx

## Agente solucionador de problemas

- Veremos cómo un agente puede encontrar una secuencia de acciones para alcanzar sus metas cuando no se logra con una sola acción.
- Un agente solucionador de problemas (*problem-solving agent*), es un agente basado en metas (*goal-based agent*).
- Usan representaciones atómicas, esto es, los estados del mundo son considerados como un todo, con una estructura interna que no es visible para el algoritmo de solución.
- Los agentes que usan una representación factorizada o estructurada más avanzada se llaman agentes planificadores.

## Meta

- Un agente inteligente debe maximizar su medida de desempeño.
- Alcanzar este objetivo algunas veces se simplifica si se adopta una meta y se trata de satisfacer.
- El primer paso para resolver un problema es establecer una meta (*Goal Formulation*).
- Lo más común es considerar una meta como un conjunto de estados del mundo (los que satisfacen la meta).
- La tarea del agente es encontrar cómo actuar, ahora y en el futuro, de tal forma que pueda alcanzar un estado meta.

## Problema

- Antes de que pueda actuar para alcanzar un estado meta, el agente necesita decidir (o nosotros necesitamos decidir por él) qué clase de acciones y estados se deben considerar.
- Formulación del problema: es el proceso de decidir qué acciones y estados considerar, dada una meta.
- El agente tiene que decidir cuál de las acciones posibles es la mejor.
- Si el agente no tiene más información, es decir, si el ambiente es desconocido, no le queda otra opción mas que seleccionar una acción al azar.

## El ambiente

- El ambiente en el que se mueve nuestro agente debe tener ciertas características para poder aplicar ciertos algoritmos:
  - Observable: El agente siempre conoce el estado actual.
  - Discreto: Para un estado dado existe solamente un número finito de acciones para escoger.
  - Conocido: Conoce cuál estado está alcanzado con cada acción.
  - Determinístico: Cada acción tiene exactamente un posible resultado.

## Solución

- Bajo esta asunciones, la solución de un problema es una secuencia fija de acciones.
- El proceso de buscar (*looking for*) por una secuencia de acciones que logren alcanzar la meta se llama búsqueda (*search*).
- Un algoritmo de búsqueda es un proceso que toma un problema como entrada y regresa una solución en la forma de una secuencia de acciones.
- Una vez que la solución es encontrada, se pueden llevar a cabo las acciones recomendadas, lo cual se conoce como fase de ejecución.

## Problemas y soluciones bien definidas

- Un problema puede ser definido formalmente por 5 componentes:
  - Estado Inicial.
    - En el que el agente inicia.
  - Descripción de las posibles acciones.
    - Dado un estado particular **s**, la función ACTIONS(s) regresa el conjunto de acciones que pueden ser ejecutadas en s.
  - Una descripción de lo que hace cada acción.
    - El nombre formal de esto es Modelo de Transición
    - Especificado por una función RESULT(s,a) que regresa el estado que resulta de aplicar la acción a al estado s.
    - Se utiliza el término sucesor para definir el estado que es alcanzable a partir de un estado, por medio de una acción simple.

## Espacio de estados

- Juntos, el estado inicial, las acciones y el modelo de transición definen implícitamente el espacio de estados de un problema, es decir, el conjunto de todos los posibles estados alcanzables a partir del estado inicial por medio de una secuencia de acciones.
- Un camino (*path*) en el espacio de estados es una secuencia de estados conectados por una secuencia de acciones.

## Los dos que faltan

- La prueba meta (*goal test*), el cual determina si un estado dado es un estado meta.
  - Algunas veces hay un conjunto de posibles estados meta y la prueba simplemente verifica si el estado es uno de ellos.
  - Algunas veces la meta es especificada por medio de una propiedad abstracta más que por una enumeración explícita del conjunto de estados.
- Una función de costo del camino que asigna un costo numérico a cada camino.
  - El agente solucionador de problemas selecciona una función de costo que refleje su propia medida de desempeño.
  - A veces se asume que el costo del camino puede ser descrito como la suma de los costos de acciones individuales a lo largo del camino.
  - El costo de un paso (*step cost*) de tomar la acción ***a*** en el estado ***s*** para alcanzar el estado ***s'*** se denota como ***c(s, a, s')***.

## Solución

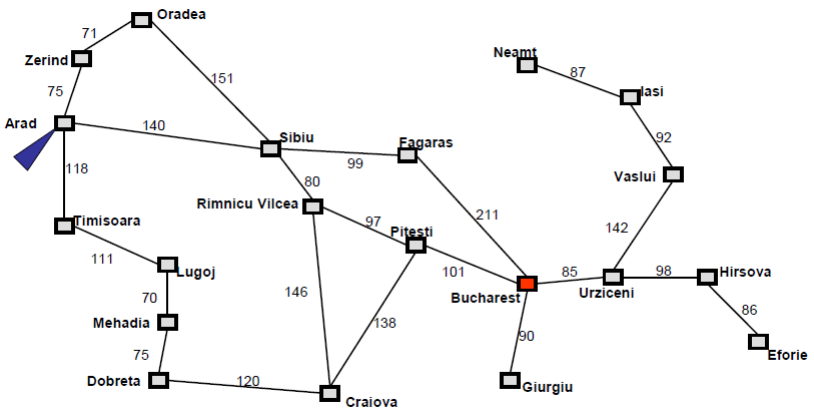
- Los elementos anteriores definen un problema y pueden colocarse juntos en una estructura de datos (o clase) la cual es dada como entrada a un algoritmo solucionador de problemas.
- Una solución a un problema es una secuencia de acciones que llevan del estado inicial a un estado meta.
- La calidad de la solución es medida por la función de costo del camino y una solución óptima tiene el menor costo de camino entre todas las soluciones posibles.

# Ejemplos de problemas

- Un tour por Rumania (Russel y Norvig).
- 8-puzzle problem (sliding-block puzzle)
- 8-Queens problem

- Objetivo: Formular el problema (definiendo los 5 puntos para cada problema)

# Un tour por Rumania

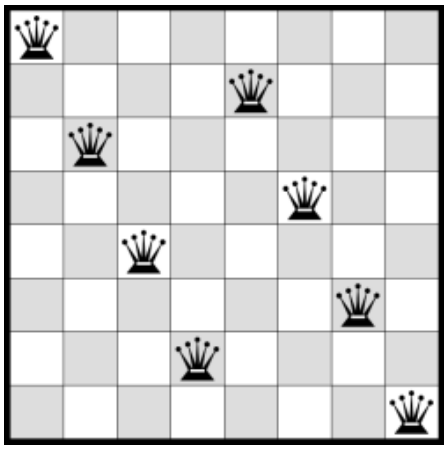


## 8-puzzle

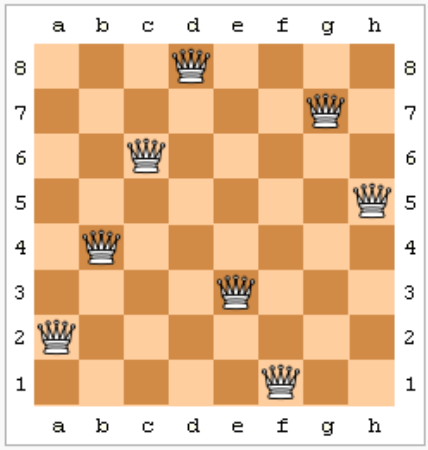
| | | |
|---|---|---|
| | 1 | 2 |
| 4 | 5 | 3 |
| 7 | 8 | 6 |

| | | |
|---|---|---|
| 1 | 2 | 3 |
| 4 | 5 | 6 |
| 7 | 8 | |

## 8-Queens



# Una solución 8-queens




Una posible solución entre las 92 posibles soluciones en un tablero de 8×8.

Fuente: https://es.wikipedia.org/wiki/Problema_de_las_ocho_reinas

# Grafo del espacio de estados

- Un grafo del espacio de estados es una representación matemática (modelo) de un problema de búsqueda:
  - Los nodos son configuraciones (abstractas) del mundo.
  - Los arcos representan sucesores (resultados de una acción).
  - La prueba meta es un conjunto de nodos meta (posiblemente uno).
- En el grafo de espacio de estados, cada nodo ocurre sólo una vez.
- Un grafo de búsqueda es la representación de cómo se está recorriendo el grafo del espacio de estados.
  - Rara vez se construye todo el grafo completo en memoria (es muy grande) pero es una idea útil.
  - Generalmente es un árbol.

# Árbol de búsqueda

- Un árbol "que pasa si" de planes y sus salidas.
- El estado inicial es el nodo raíz.
- Los hijos corresponden a los sucesores.
- Los nodos muestran estados pero corresponden a PLANES que alcanzan esos estados.
- Para muchos problemas es imposible construir el árbol completo.
- Un estado se expande aplicándole acciones legales para generar nuevos estados.
- El conjunto de todas las hojas disponibles para expansión se llama frontera.
- El proceso de expansión continúa hasta que se encuentra una solución o no se pueden expandir más estados.

# Algoritmo de búsqueda en árboles

**función** BÚSQUEDA-ÁRBOLES(*problema,frontera*) **devuelve** una solución o fallo
  *frontera* ← INSERTA(HACER-NODO(ESTADO-INICIAL[*problema*]), *frontera*)
  **hacer bucle**
    **si** VACIA?(*frontera*) **entonces devolver** fallo.
    *nodo* ← BORRAR-PRIMERO(*frontera*)
    **si** TEST-OBJETIVO[*problema*] aplicado al ESTADO[*nodo*] es cierto
      **entonces devolver** SOLUCIÓN(*nodo*)
    *frontera* ← INSERTAR-TODO(EXPANDIR(*nodo,problema*),*frontera*)

---

**función** EXPANDIR(*nodo,problema*) **devuelve** un conjunto de nodos
  *sucesores* ← conjunto vacío
  **para cada** (*acción,resultado*) **en** SUCESOR-FN[*problema*](ESTADO[*nodo*]) **hacer**
    *s* ← un nuevo NODO
    ESTADO[*s*] ← *resultado*
    NODO-PADRE[*s*] ← *nodo*
    ACCIÓN[*s*] ← *acción*
    COSTO-CAMINO[*s*] ← COSTO-CAMINO[*nodo*] + COSTO-INDIVIDUAL(*nodo,acción,s*)
    PROFUNDIDAD[*s*] ← PROFUNDIDAD[*nodo*] + 1
    añadir *s* a *sucesores*
  **devolver** *sucesores*

## Algoritmo de búsqueda en árboles

**function** TREE-SEARCH(*problem*) **returns** a solution, or failure
  initialize the frontier using the initial state of *problem*
  **loop do**
    **if** the frontier is empty **then return** failure
    choose a leaf node and remove it from the frontier
    **if** the node contains a goal state **then return** the corresponding solution
    expand the chosen node, adding the resulting nodes to the frontier

## De otra forma

1. Colocar el Estado Inicial en la Estructura.
2. Sacar un estado de la Estructura y hacerlo es Estado Actual.
3. ¿Es Estado Actual = Estado Meta? (nodo visitado)
   1. Sí, ir a FIN
4. Expandir el Estado Actual y colocar sus hijos en la Estructura.
5. Ir a 2
6. FIN

# Algoritmos de búsqueda

- Todos los algoritmos de búsqueda comparten esta estructura básica.
- Su variación consiste en cómo ellos escogen cuáles estados se expanden a continuación.
- A esto se le conoce como estrategia de búsqueda.
- En muchos casos es posible tener nodos repetidos, creando caminos redundantes.
  - La forma de evitar caminos redundantes es recordar dónde se ha estado.
  - Para hacer esto se aumenta el algoritmo de búsqueda en árboles con una estructura de datos llamada Conjunto Explorado o Nodos Visitados, la cual recuerda cada nodo expandido.
  - El nuevo algoritmo es llamado Búsqueda en Grafos.

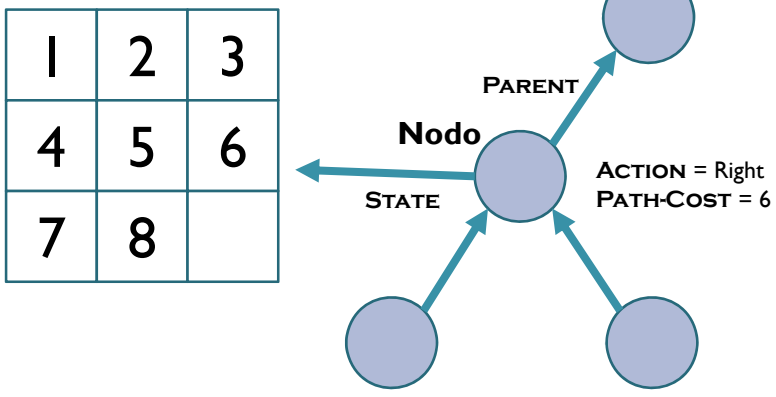
# Algoritmo de búsqueda en grafos

**function** GRAPH-SEARCH(*problem*) **returns** a solution, or failure
  initialize the frontier using the initial state of *problem*
  ***initialize the explored set to be empty***
  **loop do**
    **if** the frontier is empty **then return** failure
    choose a leaf node and remove it from the frontier
    **if** the node contains a goal state **then return** the
      corresponding solution
    ***add the node to the explored set***
    expand the chosen node, adding the resulting nodes to the frontier
      ***only if not in the frontier or explored set***

## Infraestructura para los algoritmos de búsqueda

- Los algoritmos de búsqueda requieren una estructura de datos para dar seguimiento al árbol de búsqueda que está siendo construido.
- Para cada nodo ***n*** del árbol tenemos una estructura que contiene 4 componentes:
  - n.Estado: El estado en el espacio de estados al que corresponde el nodo.
  - n. Papá: El nodo en el árbol que generó el nodo actual.
  - n.Acción: La acción que fue aplicada al papá para generar el nodo.
  - n. Costo: El costo, tradicionalmente denotado como g(n), del camino del estado inicial al nodo.

## Estructura de datos para 8-puzzle



## La función CHILD-NODE

- La función CHILD-NODE toma un nodo papá y una acción y regresa el nodo hijo resultante:

**function** CHILD-NODE(*problem*, *parent*, *action*) **returns** a node
  **return** a node with
    STATE = *probiem*.RESULT(*parent*.STATE, *action*),
    PARENT = *parent*, ACTION = *action*,
    PATH COST = *parent*,PATH-COST +
        *problem*.STEP-COST(*parent*.STATE, *action*)

## Búsqueda con un árbol

- Expandir planes potenciales (nodos del árbol)
- Mantener un *fringe* de planes parciales bajo consideración.
- Un *fringe* es el conjuntos de planes que están en la frontera del árbol (hojas).
- Tratar de expandir la menor cantidad de nodos posible.

## Medidas de desempeño de la solución de un problema

- Podemos evaluar el desempeño de un algoritmo en 4 puntos:
  - Completitud (*Completeness*): ¿El algoritmo garantiza encontrar una solución cuando exista alguna?
  - Optimalidad (*Optimality*): ¿La estrategia encuentra la solución óptima? De acuerdo a algún criterio de optimalidad.
  - Complejidad en Tiempo (*Time Complexity*): ¿Cuánto tarda encontrar una solución?
  - Complejidad en Espacio (*Space Complexity*): ¿Cuánta memoria se necesita para realizar la búsqueda?

## Complejidad

- La complejidad es expresada en términos de 3 cantidades:
  - ***b***, factor de ramificación o máximo número de sucesores de un nodo.
  - ***d***, profundidad de la meta más superficial (menos profunda, número de pasos a lo largo del camino desde la raíz).
  - ***m***, la longitud máxima de cualquier camino en el espacio de estados.
- El tiempo es normalmente medido en términos del número de nodos generados durante la búsqueda.
- El espacio en términos del máximo número de nodos almacenados en memoria.

## División

- Hasta la fecha no existe "EL" algoritmo de búsqueda que resuelva todos los problemas de forma eficiente.
- La importancia de los problemas de búsqueda ha ocasionado que, en la corta vida de las ciencias computacionales, se hayan creado una gran cantidad de algoritmos, buscando resolver eficientemente ciertos problemas.
- Existen muchas formas de clasificar la gran cantidad de algoritmos de búsqueda que existen.
- Una de las más comunes es hacerlo de acuerdo a la información del problema con la que cuentan:
  - Informados
  - No informados

## Referencia


- S. Russel and P. Norvig. Inteligencia Artificial un enfoque moderno. 2ª edición, Pearson, España (2004).