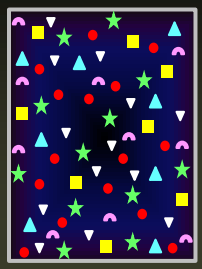
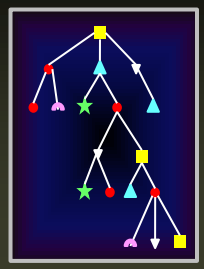

INSTITUTO POLITÉCNICO NACIONAL
Centro de Investigación en Computación
Fundamentos de Inteligencia Artificial
04
Administración de la memoria durante una búsqueda...
Dr. Salvador Godoy Calderón


La sesión anterior...

Orden de expansión…





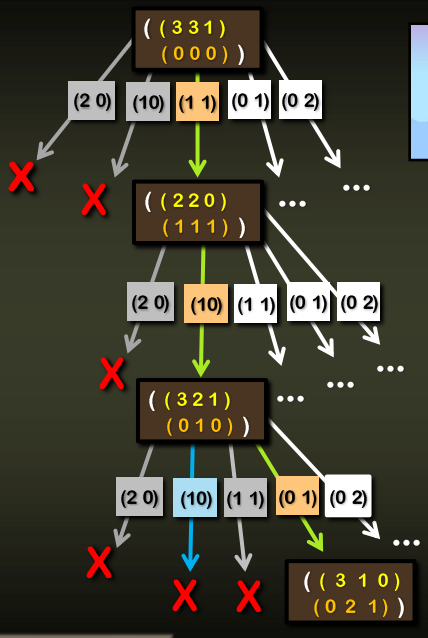
Generalmente a cada estado es posible aplicarle varios operadores.

El proceso de expansión sugiere dos posibles órdenes a seguir durante una búsqueda:

- Obtener sólo un descendiente del estado actual y proceder a expandirlo…
- Obtener todos los descendientes de cada estado y examinarlos en orden secuencial…

Dr. Salvador Godoy Calderón

A lo profundo (Depth-first)…



(2 0) (10) (1 1) (0 1) (0 2)

Orden de búsqueda:

- Mantener constante el orden de los operadores…
- Validar operadores por recursos disponibles…
- Validar estados resultantes por restricciones generales y por memoria…
- Obtener sólo el primer descendiente posible…

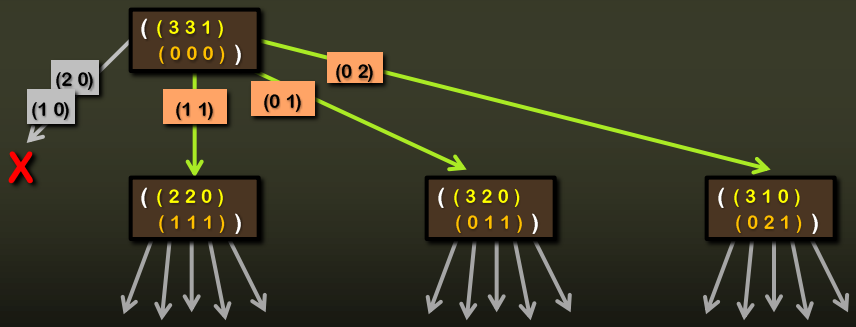
Dr. Salvador Godoy Calderón

A lo ancho (Breath-first) ...

Orden de búsqueda:

- Validar operadores y estados ...
- Obtener todos los descendientes posibles ...

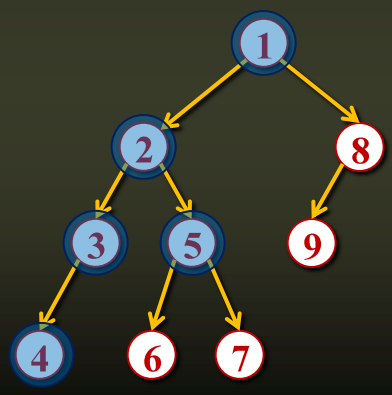
(2 0) (10) (1 1) (0 1) (0 2)



Dr. Salvador Godoy Calderón

Backtracking a lo profundo...

Al buscar a lo profundo, se extrae el siguiente nodo de la frontera de búsqueda, se generan *todos* sus descendientes y se insertan, por el inicio de la lista, en orden inverso a su creación...



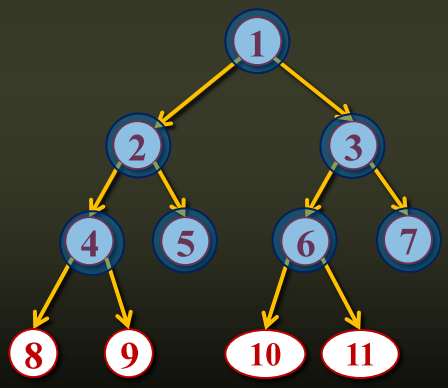
Frontera de búsqueda

- (1)
- (2 8)
- (3 5 8)
- (4 5 8)
- (5 8)
- (6 7 8)
- (7 8)
- (8)

Dr. Salvador Godoy Calderón

Backtracking a lo ancho...

Por el contrario, al buscar a lo ancho, se extrae el siguiente nodo a revisar, se generan todos sus descendientes y se insertan en el mismo orden pero por el final de la lista...



Frontera de búsqueda

(1)
(2 3)
(3 4 5)
(4 5 6 7)
(5 6 7 8 9)
(6 7 8 9)
(7 8 10 11)
(8 9 10 11)

Dr. Salvador Godoy Calderón

Algoritmo de búsqueda...

1) Definir la lista de nodos a examinar (*OPEN*), conteniendo exclusivamente el nodo con el estado inicial *s*.
2) Si *OPEN* está vacía, entonces *FRACASO* y terminar.
3) Extraer de *OPEN* el siguiente nodo (*n*).
4) *n* contiene al estado-meta?

   Sí - *ÉXITO* y reconstruir la solución

   No - Expandir el estado e insertar todos sus descendientes en *OPEN*.

Paso (2)

Dr. Salvador Godoy Calderón

Elementos globales...

```
(defparameter  *open*   '() )
(defparameter  *memory* '() )
(defparameter  *id*  0 )
```

; Frontera de búsqueda...
; Memoria de intentos previos...
; Id del siguiente nodo a crear...

```
(defparameter *ops*  '( ( :Dos-Misioneros       (2 0) )
                       ( :Un-Misionero          (1 0) )
                       ( :Misionero-y-Caníbal   (1 1) )
                       ( :Un-Caníbal            (0 1) )
                       ( :Dos-Caníbales         (0 2) ) )  )
```

```
(defun  create-node (estado  op))
(defun  barge-shore (estado)
```

método:
:depth-first
:breath-first

```
(defun insert-to-open (estado op  método))
(defun  get-from-open ( )
```


Validaciones...

```
(defun  valid-operator?  (op  estado))
(defun  valid-state? (estado)
```

```
(defun  apply-operator (op  estado))
(defun  expand (estado))
```

Búsqueda…

```
(defun blind-search (edo-inicial edo-meta método)
 (let ((nodo nil) (estado nil) (sucesores '()) (operador nil))

   (insert-to-open edo-inicial nil metodo)

   (loop for i from 1 to 10 do   ;; until (null *open*) do
         (setq nodo   (get-from-open)
               estado (second nodo)
               operador (third nodo))

         (format T "Reviso el nodo ~A ~%" nodo)

         (cond   ((equal edo-meta estado) (print "Éxito")
                                          (return))
                 (T (setq sucesores (expand estado))
                    (loop for element in sucesores do
                          (insert-to-open (first element)
                                          (second element)
                                          metodo)))))))
```


No olvidemos…

Por el momento estudiamos I.A. desde la perspectiva del Paradigma Simbólico, es decir…

Agentes que resuelven problemas mediante sistemas de símbolos y realizando búsqueda heurística…

Su grado de inteligencia depende de varios factores:

- La correctitud de la soluciones que encuentra…
- La pertinencia de las decisiones que toma…
- La calidad de la(s) soluciones que encuentra…
- La velocidad con la que encuentra soluciones…
- La flexibilidad para adaptarse a nuevos problemas…

Etc. …

Ahora…

Es necesario incluir en el agente de solución la habilidad de recordar intentos previos (memoria).

Cada estado generado puede ser descartado de su posterior análisis por alguna de las siguientes razones:

- El estado no es válido según las restricciones.
- El estado ya fue analizado previamente.

Para implementar esa memoria agregaremos otra lista (*MEMORY*) al algoritmo de búsqueda ciega ordenada…

Dr. Salvador Godoy Calderón

Mecanismo de memoria...

Para implementar correctamente la memoria de intentos previos se requiere filtrar (eliminar de la lista de descendientes) aquellos estados que previamente se hayan revisado (los que están en *MEMORY*).

(filter-memories <lista-de-estados-y-operadores>)

Elimina de la lista de estados, aquellos elementos que se encuentren en la memoria ¡que es una lista de nodos!

Recuerdos...

¿En qué momento se inserta información en la memoria?

En cuanto se extrae un nodo de la Frontera de Búsqueda, se debe ingresar a la memoria...

¿Qué se debe guradar en la memoria, nodos o estados?

Se guardan los nodos completos, ya que al final se debe reconstruir la solución...

Algoritmo...

1) Definir la lista de nodos a examinar (*OPEN*), conteniendo exclusivamente al nodo inicial. Definir la lista de nodos ya examinados (*MEMORY*) vacía.

2) Si *OPEN* está vacía, entonces *FRACASO* y terminar.

3) Extraer de *OPEN* el siguiente nodo (*n*).

4) Insertar *n* en *MEMORY*

5) *n* contiene al estado-meta ?

   Sí - *ÉXITO* y reconstruir la solución

   No - Expandir el estado, e insertar todos sus descendientes en *OPEN*.

Paso (2)

Solución...

Para resolver el problema se debe diseñar una función FILTRO. Recordemos que la estructura general de un filtro es:

```
(defun Filtra (lista prueba)
        (cond ( (null lista) nil )
              ( (funcall prueba (first lista))
                        (Filtra (rest lista) prueba) )
              ( T (cons (first lista) (Filtra (rest lista) prueba) ) ) ))
```

Necesitamos filtrar los estados descendientes del estado actual, comparándolos con la lista *memory* (que es una lista de nodos)

Es decir…

En el caso estudiado se trata de buscar algunos estados en una lista de nodos (*memory*)…

Cada estado en **descendientes** debe ser buscado en todos los nodos de *memory*…

```
>> descendientes
( (G H) (K L) … )
>> *memory*
( (0 (A B) NIL) (1 (G H) 0) (2 (K L) 1) (3 (C D) 1) … )
```

Dr. Salvador Godoy Calderón

Solución…

Primero, hagamos un predicado que indique si un estado se encuentra en algún nodo de una lista de nodos…

```
(defun remember-state? (estado memoria)
 (cond ((null memoria) Nil)
       ((equal estado (second (first memoria))) T)
       (T (remember-state? estado (rest memoria))))) )
```

```
>> *memory*
( (1 (A B) 0) (2 (G H) 1) (3 (K L) 2) )
>> (remember-state? '(G H) *memory* )
T
>> (remember-state? '(X Y) *memory* )
NIL
```

Dr. Salvador Godoy Calderón

Solución (2)…

Ahora, una función que recorra la lista de estados y cada uno lo compare con toda la lista de nodos …

```
(defun filter-memories(lista-estados-ops)
 (cond ((null lista-estados-ops) Nil)
       ((remember-state? (first (first lista-estados-ops)) *memory*)
            (filter-memories (rest lista-estados-ops)))
       (T (cons (first lista-estados-ops)
                (filter-memories (rest lista-estados-ops))))) ))
```

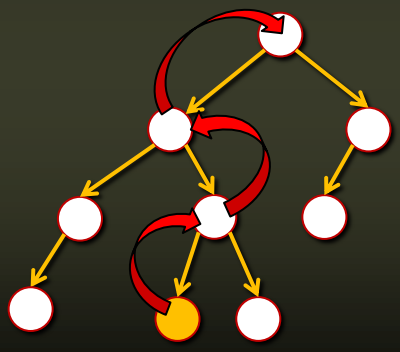

Solución (3) …

Así, la función de búsqueda puede eliminar los estados que recuerda ya haber examinado…

```
…
(setq sucesores (expand estado))
(setq sucesores (filter-memories sucesores))
…
```

Rastreo...

La solución se recupera en la memoria, rastreando los ancestros del nodo que contenía al estado meta, hasta llegar al estado inicial...



Dr. Salvador Godoy Calderón

Funciones...

```
(defun extract-solution (nodo)
  (labels ((locate-node (id lista)
             (cond ((null lista) nil)
                   ((eql id (first (first lista))) (first lista))
                   (T (locate-node id (rest lista))))))
    (let ((current (locate-node (first nodo) *memory*)))
      (loop while (not (null current)) do
        (push current *solucion*)
        (setq current (locate-node (third current) *memory*)))))
  *solucion*))
```


Funciones...

```
(defun display-solution (lista-nodos)
 (let ((nodo nil))
  (dotimes ( i (length lista-nodos))
   (setq nodo (nth i lista-nodos))
    (if (= 0 i)
          (format t " Inicio: ~A~%" (second nodo))
          (format t " ~A --> ~A~%" (fourth nodo) (second nodo)))))  )
```

Generación correcta de los nodos...

Al expandir un estado y filtrar a los descendientes, se necesita crear un nodo para cada descendiente, pero todos ellos descienden del mismo ancestro...

```
...
(setq sucesores (expand estado))
(setq sucesores (filter-memories sucesores))
(loop for element in sucesores do
      (insert-to-open (first element)
                      (second element)
                      metodo))
...
```

Dr. Salvador Godoy Calderón

Solución...

```
(defparameter  *current-ancestor*  '() )
```

```
(setq  nodo   (get-from-open)
       estado  (second  nodo)
       operador  (third nodo))
...
(setq  *current-ancestor* (first  nodo))
(setq  sucesores  (expand estado))
(setq  sucesores  (filter-memories sucesores))
(loop for  element  in  sucesores  do
     (insert-to-open  (first element)
                      (second element)
                       metodo))
...
```

Dr. Salvador Godoy Calderón

Capacidad de buscar varias veces...

```
(defun reset-all ()
   (setq *open*  nil)
   (setq *memory*  nil)
   (setq *id*  0)
   (setq *current-ancestor*  0)
   (setq *solucion* nil)  )
```

```
(defun  blind-search (edo-inicial edo-meta  metodo)
  (reset-all)
  (let ((...
```

Dr. Salvador Godoy Calderón

Búsqueda ciega...

```
(defun blind-search (edo-inicial edo-meta metodo)
 (reset-all)
 (let ((nodo nil)
       (estado nil)
       (sucesores '())
       (operador nil)
       (meta-encontrada nil))

   (insert-to-open edo-inicial nil metodo)
   (loop until (or meta-encontrada
                   (null *open*)) do
      (setq nodo (get-from-open)
            estado (second nodo)
            operador (third nodo))
      (push nodo *memory*)
      (cond ((equal edo-meta estado)
             (format t "Éxito. Meta encontrada en ~A pasos~%" (first nodo))
             (display-solution (extract-solution nodo))
             (setq meta-encontrada T))

            (t (setq *current-ancestor* (first nodo))
               (setq sucesores (expand estado))
               (setq sucesores (filter-memories sucesores))
               (loop for element in sucesores do
                (insert-to-open (first element) (second element) metodo))))) )
```


Efecto...

La solución se despliega de forma simbólica...

```
CL-USER> (blind-search INICIAL META :depth-first)
Éxito. Meta encontrada en 15 pasos
 Inicio: ((3 3 1) (0 0 0))
 MISIONERO-Y-CANÍBAL -->  ((2 2 0) (1 1 1))
 UN-MISIONERO -->  ((3 2 1) (0 1 0))
 DOS-CANÍBALES -->  ((3 0 0) (0 3 1))
 UN-CANÍBAL -->  ((3 1 1) (0 2 0))
 DOS-MISIONEROS -->  ((1 1 0) (2 2 1))
 MISIONERO-Y-CANÍBAL -->  ((2 2 1) (1 1 0))
 DOS-MISIONEROS -->  ((0 2 0) (3
 UN-CANÍBAL -->  ((0 3 1) (3 0 0))
 DOS-CANÍBALES -->  ((0 1 0) (3 2
 UN-MISIONERO -->  ((1 1 1) (2 2
 MISIONERO-Y-CANÍBAL -->  ((0 0
NIL
```

```
CL-USER> (blind-search INICIAL META :breath-first)
Éxito. Meta encontrada en 24 pasos
 Inicio: ((3 3 1) (0 0 0))
 DOS-CANÍBALES -->  ((3 1 0) (0 2 1))
 UN-CANÍBAL -->  ((3 2 1) (0 1 0))
 DOS-CANÍBALES -->  ((3 0 0) (0 3 1))
 UN-CANÍBAL -->  ((3 1 1) (0 2 0))
 DOS-MISIONEROS -->  ((1 1 0) (2 2 1))
 MISIONERO-Y-CANÍBAL -->  ((2 2 1) (1 1 0))
 DOS-MISIONEROS -->  ((0 2 0) (3 1 1))
 UN-CANÍBAL -->  ((0 3 1) (3 0 0))
 DOS-CANÍBALES -->  ((0 1 0) (3 2 1))
 UN-CANÍBAL -->  ((0 2 1) (3 1 0))
 DOS-CANÍBALES -->  ((0 0 0) (3 3 1))
NIL
```

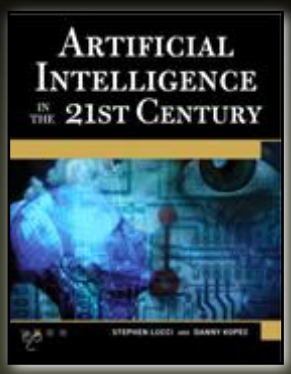
Además...

Para estudiar la eficiencia de los métodos de búsqueda se requieren estadísticas de cada problema solucionado:

Nodos creados: 310
Nodos expandidos: 179
Longitud máxima de la Frontera de búsqueda: 18
Longitud de la solución: 11 operadores
Tiempo para encontrar la solución: 2.45 segundos

A partir de esta sesión se deberán calcular todos estos indicadores cada vez que se resuelva un problema mediante búsqueda...

Estudio...



Del libro de Lucci - Kopec,

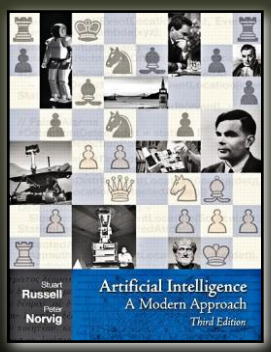
Part II: *Fundamentals*

Sec. 2 *Uninformed Search Intelligence*

Del libro de Russell-Norvig,

Cap. 3 *Solving problems by searching*

Sec. 3.1 - **Sec. 3.4**




¡ Gracias !
Dr. Salvador Godoy Calderón
CIC - IPN