



## Fundamentos de Inteligencia Artificial

03



### Memoria de intentos previos...

Dr. Salvador Godoy Calderón

*La sesión anterior...*



*Orden de expansión...*

Dos órdenes a seguir durante una búsqueda:

- ◆ Buscar siguiendo las ramas del árbol de búsqueda (a lo profundo, algoritmo *DFS Depth-First Search*)
- ◆ Buscar siguiendo los niveles del árbol de búsqueda (a lo ancho, algoritmo *BFS Breath-First Search*)

Dr. Salvador Godoy Calderón

3

*A lo profundo (Depth-first)...*

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

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

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

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

Estrategia de búsqueda:

- ◆ Mantener constante el orden de los operadores...
- ◆ Revisar los tres filtros posibles...

Dr. Salvador Godoy Calderón

4

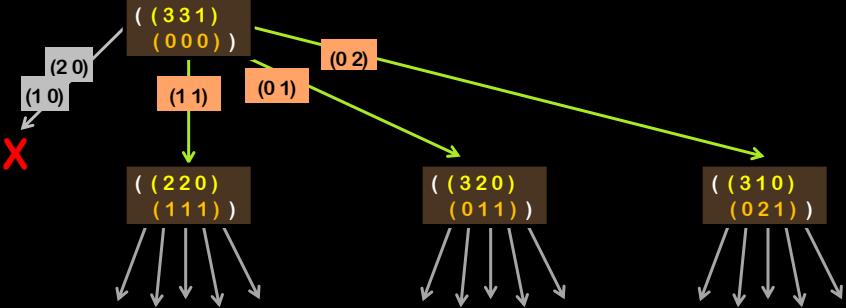
 **A lo ancho (Breath-first) ...**

Estrategia de búsqueda:

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



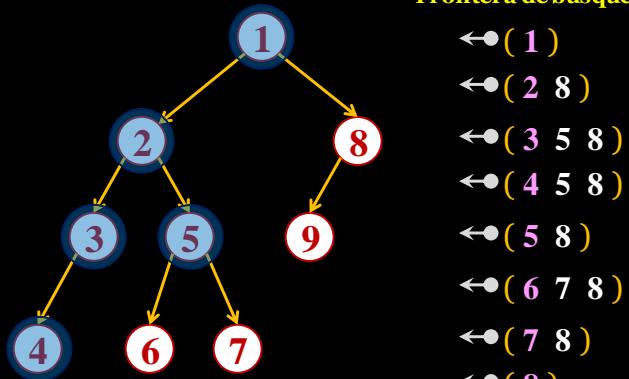
(2 0) (10) (1 1) (0 1) (0 2)



Dr. Salvador Godoy Calderón 5

 **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 6

**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**

- $\leftarrow \bullet (1)$
- $\leftarrow \bullet (2 \ 3)$
- $\leftarrow \bullet (3 \ 4 \ 5)$
- $\leftarrow \bullet (4 \ 5 \ 6 \ 7)$
- $\leftarrow \bullet (5 \ 6 \ 7 \ 8 \ 9)$
- $\leftarrow \bullet (6 \ 7 \ 8 \ 9)$
- $\leftarrow \bullet (7 \ 8 \ 10 \ 11)$
- $\leftarrow \bullet (8 \ 9 \ 10 \ 11)$

```

graph TD
    1((1)) --> 2((2))
    1((1)) --> 3((3))
    2((2)) --> 4((4))
    2((2)) --> 5((5))
    3((3)) --> 6((6))
    3((3)) --> 7((7))
    4((4)) --> 8((8))
    4((4)) --> 9((9))
    6((6)) --> 10((10))
    6((6)) --> 11((11))
  
```

Dr. Salvador Godoy Calderón

7

**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) De lo contrario, extraer de  $OPEN$  el siguiente nodo ( $n$ ).
- 4)  $n$  contiene al estado-meta?

*Sí* - *ÉXITO* y reconstruir la solución

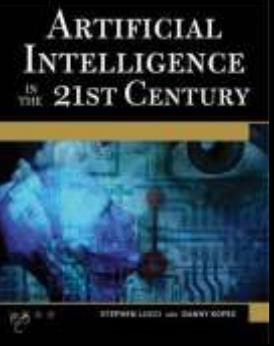
Paso (2)

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

Dr. Salvador Godoy Calderón

8

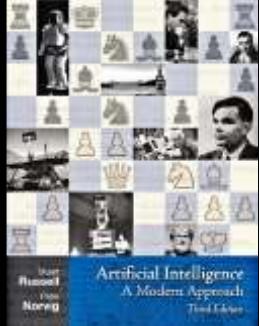
 *Estudio...*



ARTIFICIAL  
INTELLIGENCE  
IN THE  
21ST CENTURY

STEPHAN LUCCI AND DANNY KOPEC

Del libro de Lucci - Kopec,  
**Part II: Fundamentals**  
**Sec. 2** *Uninformed Search Intelligence*



Stuart  
Russell  
Peter  
Norvig

Artificial Intelligence  
A Modern Approach  
Third Edition

Del libro de Russell-Norvig,  
**Cap. 3** *Solving problems by searching*  
**Sec. 3.1 - Sec. 3.4**

Dr. Salvador Godoy Calderón

9



**Falta algo...**

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 dos 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

11

**Rastreo de la solución...**

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

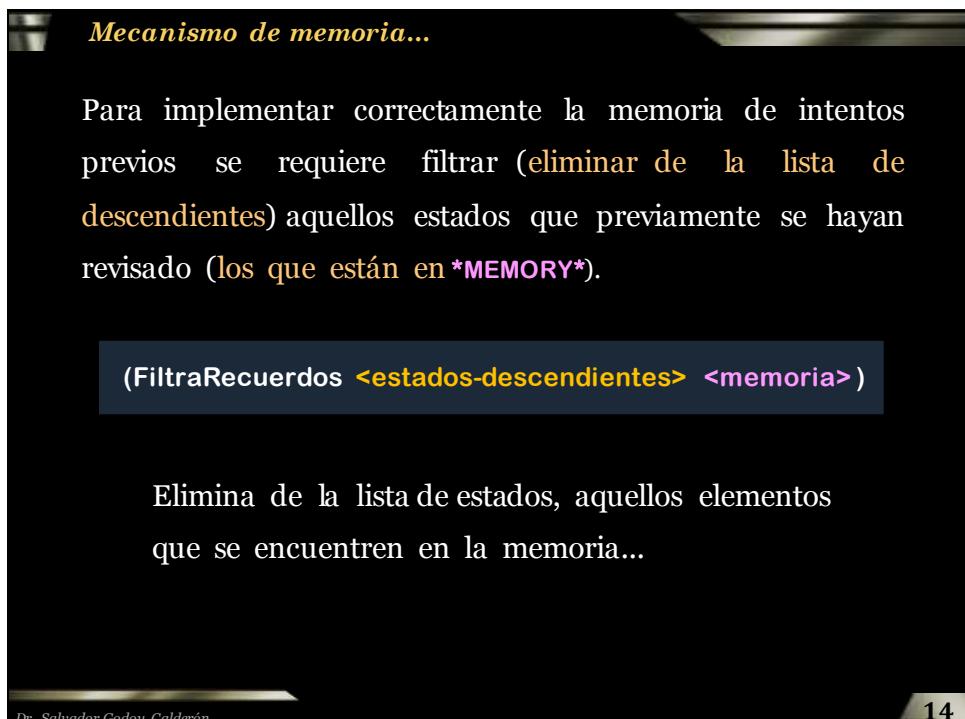
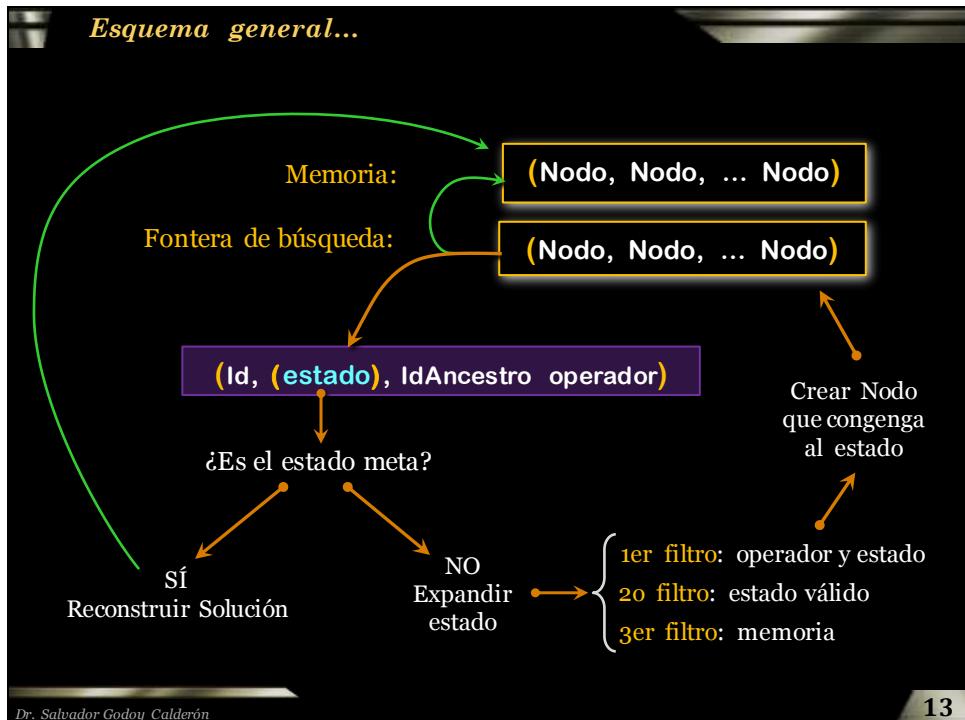
```

graph TD
    Root(( )) --> N1(( ))
    Root --> N2(( ))
    N1 --> N3(( ))
    N1 --> N4(( ))
    N3 --> N5(( ))
    N3 --> N6(( ))
    N5 --> Goal(( ))
    N5 --> N7(( ))
    N6 --> N8(( ))
    N6 --> N9(( ))
    style Goal fill:#ffff00,stroke:#ff0000,stroke-width:2px
    style N1 fill:#ffff00,stroke:#ff0000,stroke-width:2px
    style N3 fill:#ffff00,stroke:#ff0000,stroke-width:2px
    style N5 fill:#ffff00,stroke:#ff0000,stroke-width:2px
    style N7 fill:#ffff00,stroke:#ff0000,stroke-width:2px
    style N9 fill:#ffff00,stroke:#ff0000,stroke-width:2px
    style N2 fill:#ffff00,stroke:#ff0000,stroke-width:2px
    style N4 fill:#ffff00,stroke:#ff0000,stroke-width:2px
    style N6 fill:#ffff00,stroke:#ff0000,stroke-width:2px
    style N8 fill:#ffff00,stroke:#ff0000,stroke-width:2px
    style N10 fill:#ffff00,stroke:#ff0000,stroke-width:2px
  
```

The diagram illustrates a search tree with nodes represented as circles. The root node is white. It has two children, both of which are white. The left child's left child is also white. The left child's right child has two white children of its own. The bottom-most node in this subtree is highlighted with a yellow fill and a red border, indicating it is the goal state. Red curved arrows point from the goal node up through its parents to the root node, tracing the path of the solution.

Dr. Salvador Godoy Calderón

12



 *Recuerdos...*

La lista **\*MEMORY\*** se usa para almacenar los **NODOS** ya revisados durante la búsqueda...

Al generar un nuevo estado se debe revisar si es un estado **meta**, si es válido, o si se puede **recordar** haberlo revisado previamente (buscarlo en **\*MEMORY\***)...

Pero ambas comparaciones deben tomar en cuenta sólo el **estado del sistema**, no el nodo completo...

Dr. Salvador Godoy Calderón 15

 *Solución...*

Para resolver el problema se debe diseñar una función **FILTRO**:

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

Filtrar los estados descendientes que se encuentren en la lista **\*MEMORY\*** (que es una lista de nodos)

Dr. Salvador Godoy Calderón 16

 *Sin embargo...*

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) (M N) ... )

>> *MEMORY*
( (0 (A B) NIL) (1 (G H) 0) (2 (K L) 1) (3 (C D) 1) ... )
```

Dr. Salvador Godoy Calderón 17

 *Solución...*

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

```
(defun Recuerdo? (estado listanodos)
  (cond ((null listanodos) NIL)
        ((equal estado (second (first listanodos))) T)
        (T (Recuerdo? estado (rest listanodos)))))
```

```
>> *MEMORY*
( (1 (A B) 0) (2 (G H) 1) (3 (K L) 2) )
>> (Recuerdo? '(G H) *MEMORY*)
T
>> (Recuerdo? '(X Y) *MEMORY*)
NIL
```

Dr. Salvador Godoy Calderón 18

 *Solución (2)...*

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

```
(defun FiltraRecuerdos (listaestados listanodos)
  (cond ((null listaestados) NIL)
    ((null listanodos) listaestados)
    ((Recuerdo? (first listaestados) listanodos)
      (FiltraRecuerdos (rest listaestados) listanodos))
    (T (cons (first listaestados)
      (FiltraRecuerdos (rest listaestados) listanodos))))))
```

Es global...

Dr. Salvador Godoy Calderón

19

 *Solución (3) ...*

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

```
...
(setq sucesores (Expande (EstadoEn nodo)))
(setq sucesores (FiltraRecuerdos sucesores *MEMORY*))
...
```

Dr. Salvador Godoy Calderón

20

*Algoritmo de búsqueda...*

```
(defun BúsquedaCiega (edoInicial edoMeta método)
  (let ((nodo NIL) (edo NIL) (sucesores '()) (IdAncestro -1) (op -1))
    (InsertaNodo (CreaNodo *Id* edoInicial -1 NIL) *OPEN* método)
    (loop until (null *OPEN*) do
      (setq nodo (ExtraeNodo *OPEN* método))
      (InsertaNodo nodo *MEMORY* método)
      (cond
        ((equal edoMeta (EstadoEn nodo)) → En CLOSED no
         importa el orden...
          (RastreaSolución nodo *MEMORY*)
          (return) )
        (T (setq edo (EstadoEn nodo))
           (setq sucesores (Expande edo)))
           (setq sucesores (FiltraRecuerdos sucesores *MEMORY*))
           (dolist (n sucesores)
             → (<obtener parámetros de nuevo nodo>
                (InsertaNodo (CreaNodo *Id* <parámetros de nodo>
                                         *OPEN* método)))))))
```

Dr. Salvador Godoy Calderón

21

*Funciones...*

```
(defun extract-solution (nodo)
  (labels ((locate-node (id lista) ; función local de búsqueda
            (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*))
```

Dr. Salvador Godoy Calderón

22

**Funciones...**

```
(defun display-solution (lista-nodos)
  "Despliega la solución en forma conveniente y numerando los pasos"
  (format t "Solución con ~A pasos:~%~%" (1- (length lista-nodos)))
  (let ((nodo nil))
    (dotimes (i (length lista-nodos))
      (setq nodo (nth i lista-nodos))
      (if (= i 0)
          (format t "Inicio en: ~A~%" (second nodo))
          (format t "\(~2A\)\ aplicando ~20A se llega a ~A~%" i
                  (fourth nodo) (second nodo))))))
```

Dr. Salvador Godoy Calderón

23

**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 ((eql edo-meta estado)
                   (format t "Exito. 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))))))) )
```

Dr. Salvador Godoy Calderón

24

Solución...

```

CL-USER> (load "Misioneros-Canibales.lisp")
T
CL-USER> (blind-search '((3 3 1)(0 0 0)) '((0 0 0)(3 3 1)) :depth-first)
Éxito. Meta encontrada en 15 intentos
Solución con 11 pasos:

Inicio en: ((3 3 1) (0 0 0))
(1 ) aplicando MISIONERO-Y-CANIBAL se llega a ((2 2 0) (1 1 1))
(2 ) aplicando UN-MISIONERO se llega a ((3 2 1) (0 1 0))
(3 ) a
(4 ) a
(5 ) a
(6 ) a
(7 ) a
(8 ) a
(9 ) a
(10) a
NIL
CL-USER> (blind-search '((3 3 1)(0 0 0)) '((0 0 0)(3 3 1)) :breath-first)
Éxito. Meta encontrada en 24 intentos
Solución con 11 pasos:

Inicio en: ((3 3 1) (0 0 0))
(1 ) aplicando DOS-CANIBALES se llega a ((3 1 0) (0 2 1))
(2 ) aplicando UN-CANIBAL se llega a ((3 2 1) (0 1 0))
(3 ) aplicando DOS-CANIBALES se llega a ((3 0 0) (0 3 1))
(4 ) aplicando UN-CANIBAL se llega a ((3 1 1) (0 2 0))
(5 ) aplicando DOS-MISIONEROS se llega a ((1 1 0) (2 2 1))
(6 ) aplicando MISIONERO-Y-CANIBAL se llega a ((2 2 1) (1 1 0))
(7 ) aplicando DOS-MISIONEROS se llega a ((0 2 0) (3 1 1))
(8 ) aplicando UN-CANIBAL se llega a ((0 3 1) (3 0 0))
(9 ) aplicando DOS-CANIBALES se llega a ((0 1 0) (3 2 1))
(10) aplicando UN-CANIBAL se llega a ((0 2 1) (3 1 0))
(11) aplicando DOS-CANIBALES se llega a ((0 0 0) (3 3 1))
NIL
CL-USER>

```

Dr. Salvador Godoy Calderón

25



## Indicadores de desempeño...

**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...

Dr. Salvador Godoy Calderón

27

**Primera opción...**

Usar la función **TIME** de *LISP*...

```
>> (time (blind-search INICIO META :depth-first))

Éxito. Meta encontrada en 15 intentos
Solución con 11 pasos
Inicio en: ((3 3 1) (0 0 0))
(1) aplicando MISIONERO-Y-CANIBAL se llega a ((2 2 0) (1 1 1))
(2) aplicando UN-MISIONERO se llega a ((3 2 1) (0 1 0))
(3) aplicando DOS-CANIBALES se llega a ((3 0 0) (0 3 1))
(4) aplicando UN-CANIBAL se llega a ((3 1 1) (0 2 0))
(5) aplicando DOS-MISIONEROS se llega a ((1 1 0) (2 2 1))
(6) aplicando MISIONERO-Y-CANIBAL se llega a ((2 2 1) (1 1 0))
(7) aplicando DOS-MISIONEROS se llega a ((0 2 0) (3 1 1))
(8) aplicando UN-CANIBAL se llega a ((0 3 1) (3 0 0))
(9) aplicando DOS-CANIBALES se llega a ((0 1 0) (3 2 1))
(10) aplicando UN-MISIONERO se llega a ((1 1 1) (2 2 0))
(11) aplicando MISIONERO-Y-CANIBAL se llega a ((0 0 0) (3 3 1))

Evaluation took:
  0.001 seconds of real time
  0.000000 seconds of total run time (0.000000 user, 0.000000 system)
  0.00% CPU
  799,673 processor cycles
  65,264 bytes consed
```

Dr. Salvador Godoy Calderón

28

**Segunda opción...**

*Common LISP* trabaja con dos tipos de tiempo:

- ◆ **Tiempo Universal:** el “tiempo real” que transcurre en nuestro plano de existencia. Puede ser codificado o decodificado.
- ◆ **Tiempo Interno:** el tiempo que transcurre dentro del CPU. Se usa para medir tiempos de ejecución de algún proceso.

Dr. Salvador Godoy Calderón 29

**Tiempo de ejecución...**

Se mide de acuerdo al reloj interno de la computadora (oscilador) en **Ticks de reloj...**

**>> internal-time-units-per-second**  
**1000**

**(GET-INTERNAL-REAL-TIME )**  
 Leer el tiempo real codificado como número entero...

**(GET-INTERNAL-RUN-TIME )**  
 Leer el número de ticks del reloj interno...

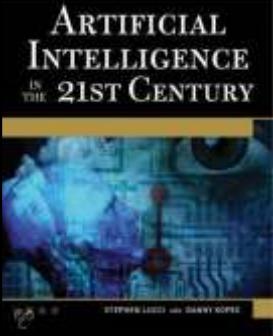
Dr. Salvador Godoy Calderón 30

**Estrategia...**

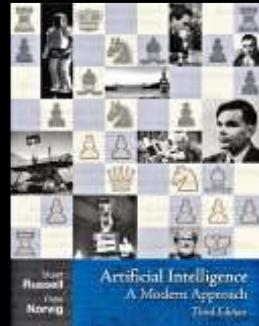
- 1) Tomar el tiempo inicial con **get-internal-run-time**  
**Tiempo1**
- 2) Ejecutar proceso (el que se quiere evaluar)
- 3) Volver a medir tiempo con **get-internal-run-time**  
**Tiempo2**
- 4) Calcular  $\frac{\text{Tiempo2} - \text{Tiempo1}}{\# \text{ ticks por segundo}} \text{ internal-time-units-per-second}$

Dr. Salvador Godoy Calderón 31

**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**

Dr. Salvador Godoy Calderón 32

**TAREA...**

- 1) Reconstruir y probar el programa para resolver el problema de misioneros y caníbales...
- 2) Siguiendo la misma metodología, programar la solución para los problemas de:
  - ◆ Ranas que cruzan el estanque...
  - ◆ Granjero, lobo, oveja y legumbres (*GLOL*)

*Dr. Salvador Godoy Calderón*

33



**TAREA...**

Los dos programas deben cumplir las siguientes condiciones:

- 1) Usar el algoritmo generalizado para buscar tanto a lo profundo como a lo ancho...
- 2) Calcular los 5 indicadores de desempeño...
- 3) Desplegar la solución en el mismo formato que el ejemplo de Misioneros y Caníbales...
- 4) Estar “adecuadamente” comentado...

- ◆ NO OLVIDE subir su tarea en formato de archivo de texto *Linux (UTF-8)*...

*Dr. Salvador Godoy Calderón*

34

