PONTIFICIA
UNIVERSIDAD
CATÓLICA
DEL PERÚ

INTELIGENCIA ARTIFICIAL (INF371)

RESOLUCIÓN DE PROBLEMAS CON BÚSQUEDA: ESTRATEGIAS DE BÚSQUEDA CIEGA

Dr. Edwin Villanueva Talavera

Contenido

- Estrategias de Búsqueda sin Información
  - Búsqueda en amplitud (*Breadth-first search*)
  - Búsqueda en profundidad (*Depth-first search*)
  - Búsqueda de costo uniforme
  - Búsqueda en profundidad limitada
  - Búsqueda de profundización iterativa
  - Búsqueda bidireccional

Bibliografía:

Capitulo 3.3 y 3.4 del libro:

Stuart Russell & Peter Norvig "Artificial Intelligence: A modern Approach", Prentice Hall, Third Edition, 2010

Estrategias de Búsqueda

Recordando: pasos de un agente básico de resolucion de problemas

- Formulación de objetivo
- Formulación de problema:
  - Estado inicial, espacio de estados, acciones, modelo de transición, costo de camino
- Búsqueda de solución:
  - encuentra una secuencia de acciones para llegar a un estado objetivo
- Ejecución de solución

Estrategias de Búsqueda

Estrategia general de **búsqueda sin memoria de estados visitados** (busqueda en árbol)

```
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
```

Estrategias de Búsqueda

Estrategia general de **búsqueda con memoria de estados visitados** (busqueda en grafo)

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

Estrategias de Búsqueda

Búsqueda sin información o búsqueda ciega

- Estrategias de búsqueda sin información usan solamente la información disponible en la definición del problema
  - Solo generan sucesores verificando si es estado objetivo
- Las estrategias de búsqueda sin información se distinguen por la orden en que los nodos son expandidos.
  - Búsqueda en amplitud (*Breadth-first search*)
  - Búsqueda en profundidad (*Depth-first search*)
  - Búsqueda de costo uniforme
  - Búsqueda en profundidad limitada
  - Búsqueda de profundización iterativa
  - Búsqueda bidireccional

Estrategias de Búsqueda

Evaluación de desempeño

- Estrategias son evaluadas de acuerdo a los siguientes criterios
  - Completitud: el algoritmo siempre encuentra la solución?
  - Complejidad de tiempo: número de nodos generados
  - Complejidad de espacio: número máximo de nodos en memoria
  - Optimalidad: la estrategia encuentra la solución optima?
    - Una solución óptima es una solución con menor costo de camino.
- Complejidad de tiempo y espacio son medidos en función de:
  - *b*: máximo factor de ramificación del árbol (numero máximo de sucesores de cualquier nodo)
  - *d*: profundidad del nodo objetivo menos profundo
  - *m:* tamaño máximo de cualquier camino en el espacio de estados

Búsqueda en amplitud

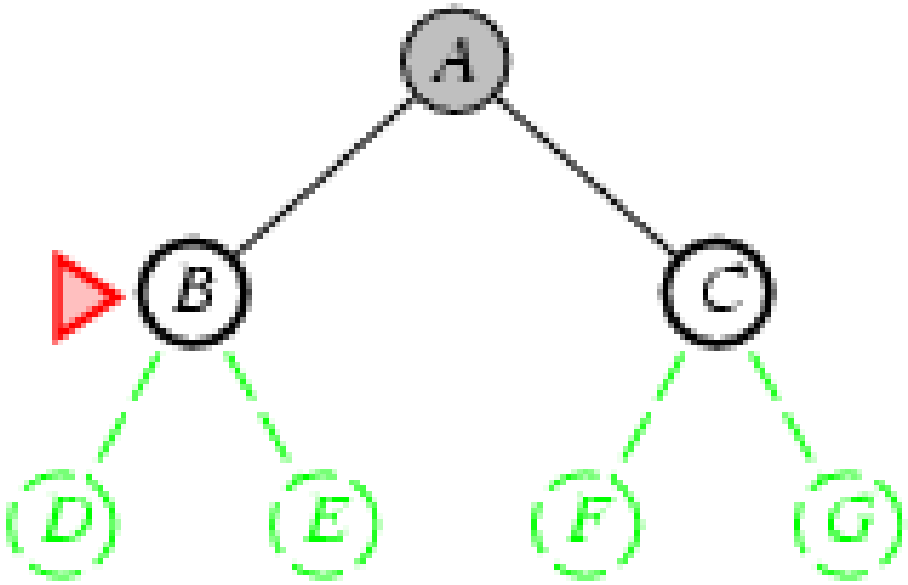
- Expandir el nodo aun no expandido mas cerca de la raíz
- Implementación: Puede ser TREE-SEARCH o GRAPH-SEARCH usando como frontera una cola FIFO:

```
function BREADTH-FIRST-SEARCH(problem) returns a solution, or failure

    node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
    if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
    frontier ← a FIFO queue with node as the only element
    explored ← an empty set
    loop do
        if EMPTY?(frontier) then return failure
        node ← POP(frontier)   /* chooses the shallowest node in frontier */
        add node.STATE to explored
        for each action in problem.ACTIONS(node.STATE) do
            child ← CHILD-NODE(problem, node, action)
            if child.STATE is not in explored or frontier then
                if problem.GOAL-TEST(child.STATE) then return SOLUTION(child)
                frontier ← INSERT(child, frontier)
```

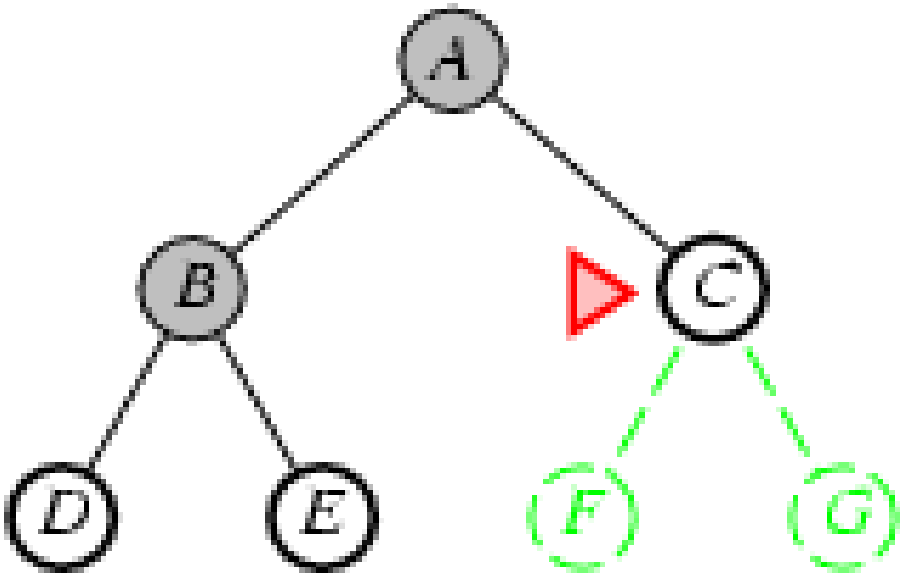
Búsqueda en amplitud

Búsqueda en Amplitud: ejemplo de exploración de nodos



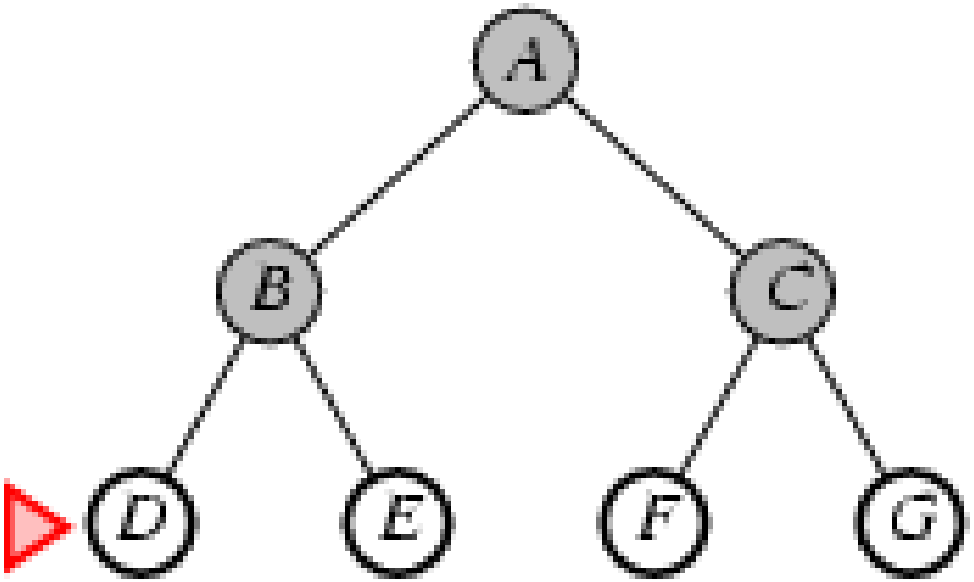
Búsqueda en amplitud

Búsqueda en Amplitud: ejemplo de exploración de nodos



Búsqueda en amplitud

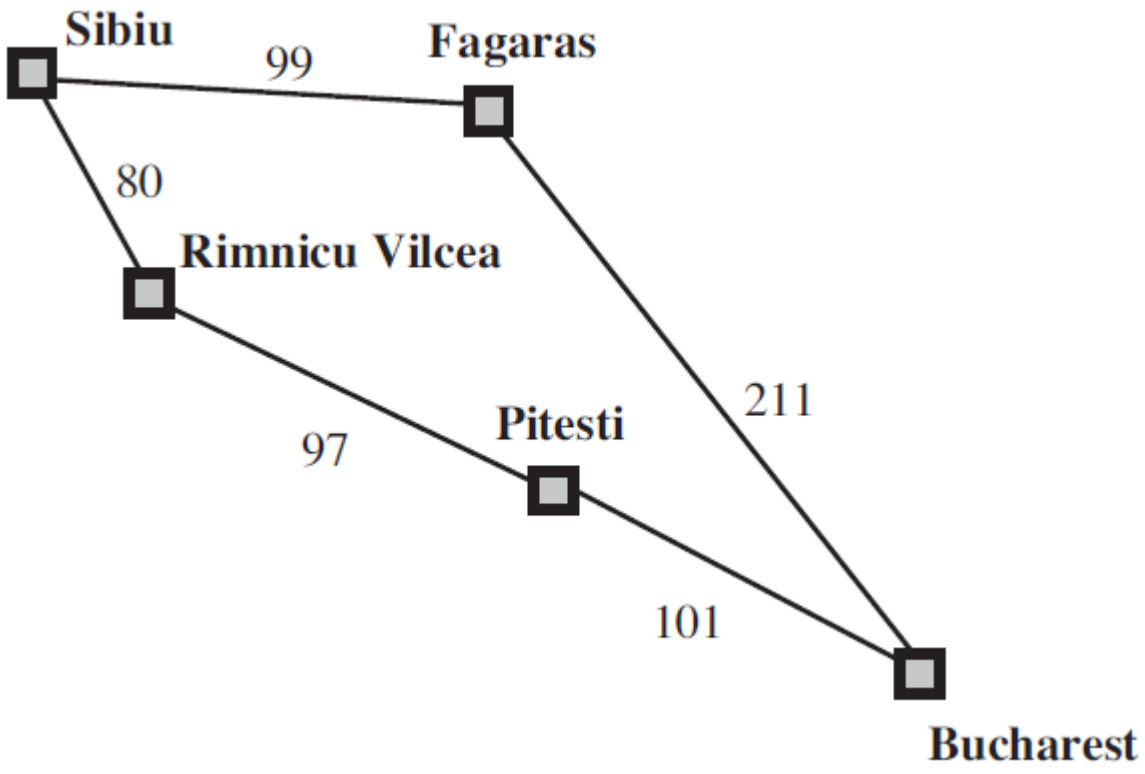
Búsqueda en Amplitud: ejemplo de exploración de nodos



Búsqueda en amplitud

Búsqueda en amplitud (Ejercicio)

- Aplicar búsqueda en amplitud en el mapa de Rumania para llegar a *Bucharest* partiendo de *Sibiu*



Búsqueda en amplitud

Propiedades de Búsqueda en amplitud

- Completa? SI, si b es finito
- Complejidad de tiempo:

  $1+b+b^2+b^3+\ldots+b^d = O(b^d)$ (impl. BFS)

  $1+b+b^2+b^3+\ldots+b^d + b(b^d) = O(b^{d+1})$ (impl. Graph-Search)
- Complejidad de espacio:
  - Existe $O(b^{d-1})$ nodos en *explored set* y $O(b^d)$ en la frontera, asi que la complejidad espacial es dominada por la frontera: $O(b^d)$
- Optima? SI, si todos las acciones tuvieran los mismos costos

Búsqueda en amplitud

Propiedades de Búsqueda en amplitud

- Con un factor de ramificación b=10 y suponiendo que puedan ser generados 1 millón de nodos por segundo y que cada nodo requiera 1KB de espacio, se tendría:

Depth	Nodes	Time	Memory
2	110	.11 milliseconds	107 kilobytes
4	11,110	11 milliseconds	10.6 megabytes
6	10^6	1.1 seconds	1 gigabyte
8	10^8	2 minutes	103 gigabytes
10	10^{10}	3 hours	10 terabytes
12	10^{12}	13 days	1 petabyte
14	10^{14}	3.5 years	99 petabytes
16	10^{16}	350 years	10 exabytes

Búsqueda en profundidad

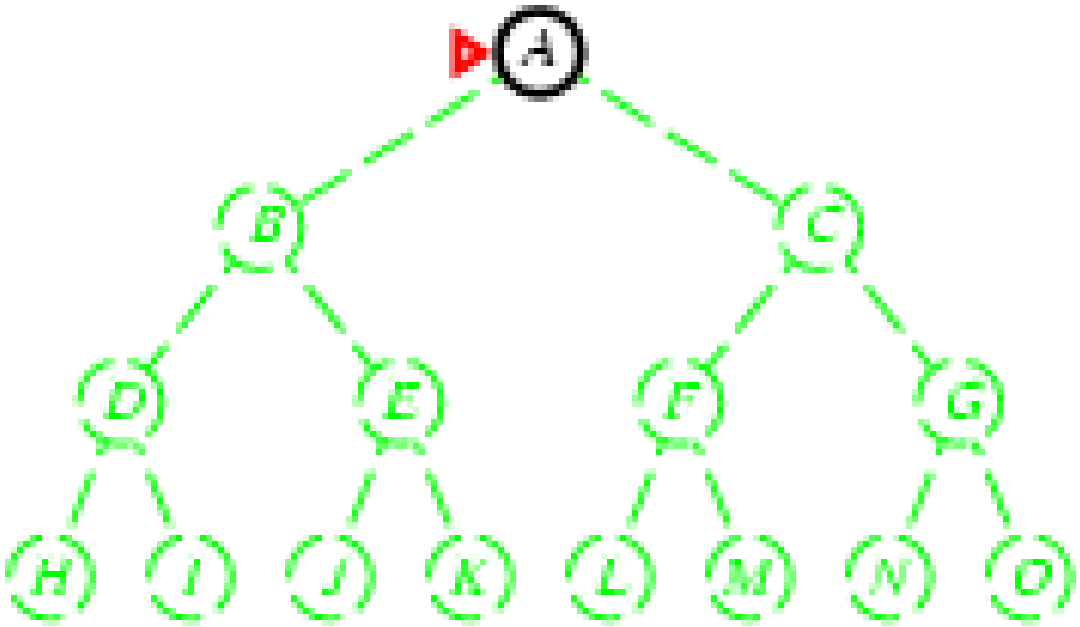
- Expande el nodo no expandido mas profundo
- Implementación: Puede ser GRAPH-SEARCH usando como frontera una lista LIFO (last-in, first-out), también conocida como pila:

```
function DEPTH-FIRST-SEARCH(problem) returns a solution, or failure

  node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
  frontier ← a LIFO list (stack) with node as the only element
  explored ← an empty set
  loop do
      if EMPTY?(frontier) then return failure
      node ← POP(frontier)  // chooses the most recent node in frontier
      if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
      add node.STATE to explored
      for each action in problem.ACTIONS(node.STATE) do
          child ← CHILD-NODE(problem, node, action)
          if child.STATE is not in explored or frontier then
              frontier ← INSERT(child, frontier)
```

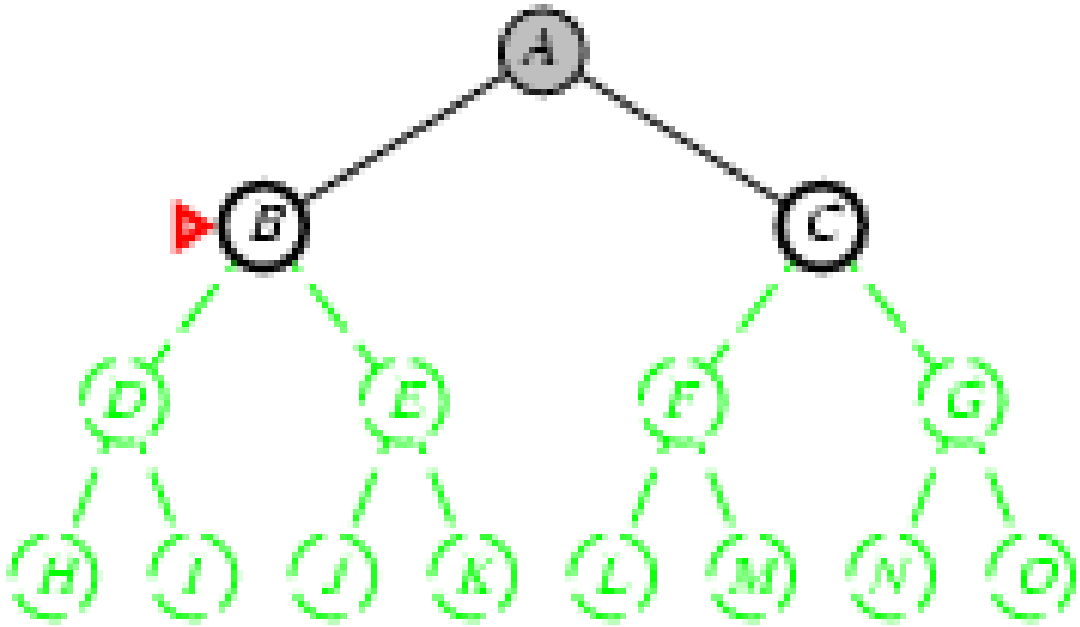
Búsqueda en profundidad

Ejemplo de exploración de nodos en búsqueda en profundidad



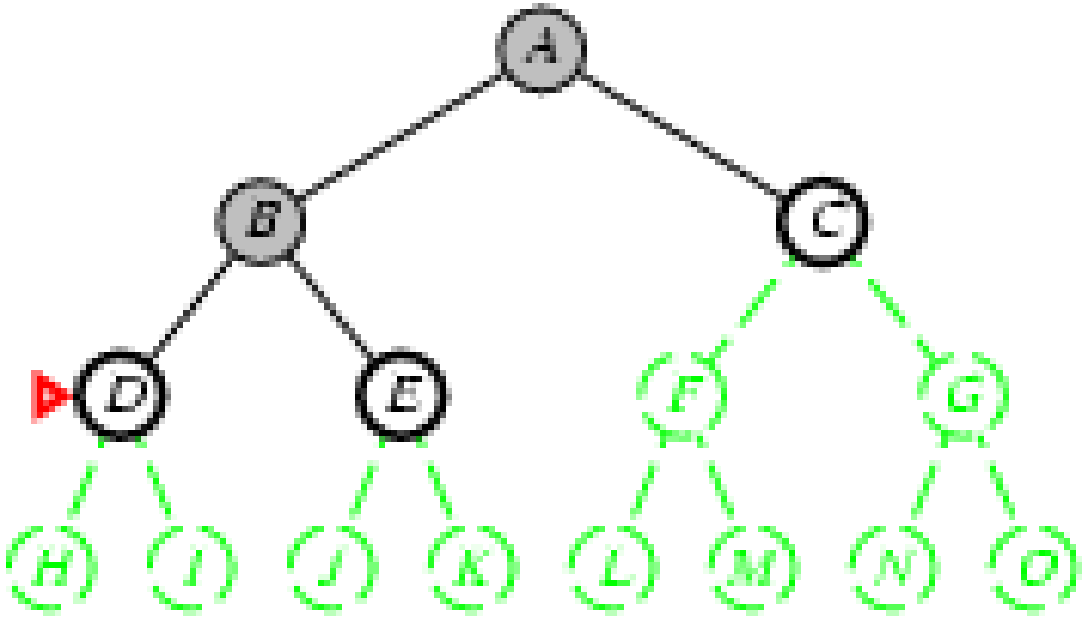
Búsqueda en profundidad

Ejemplo de exploración de nodos en búsqueda en profundidad



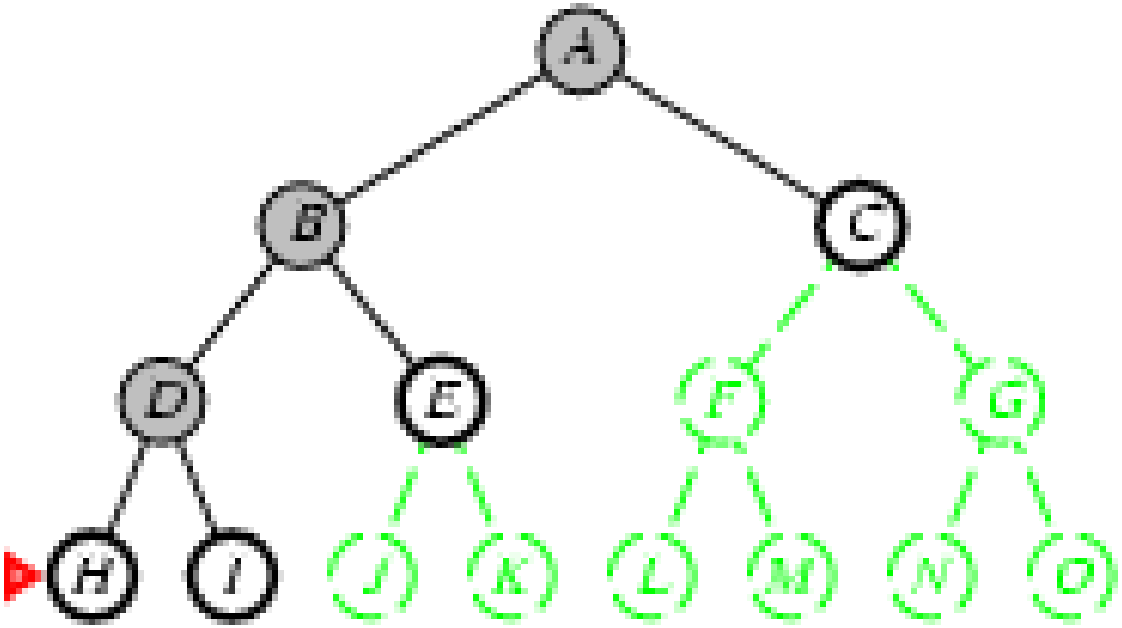
Búsqueda en profundidad

Ejemplo de exploración de nodos en búsqueda en profundidad



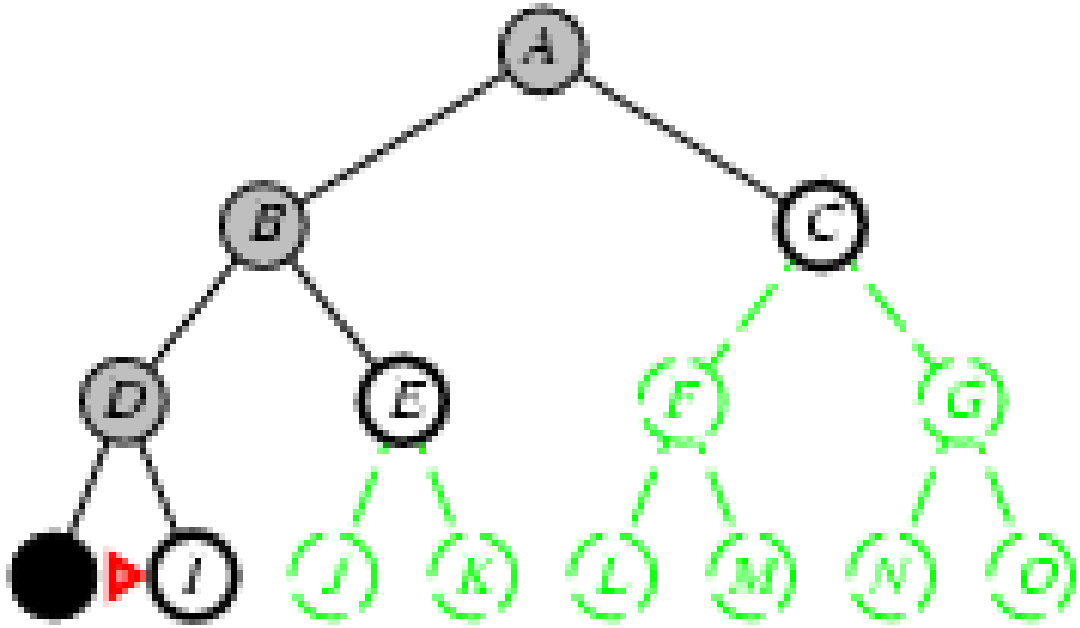
Búsqueda en profundidad

Ejemplo de exploración de nodos en búsqueda en profundidad



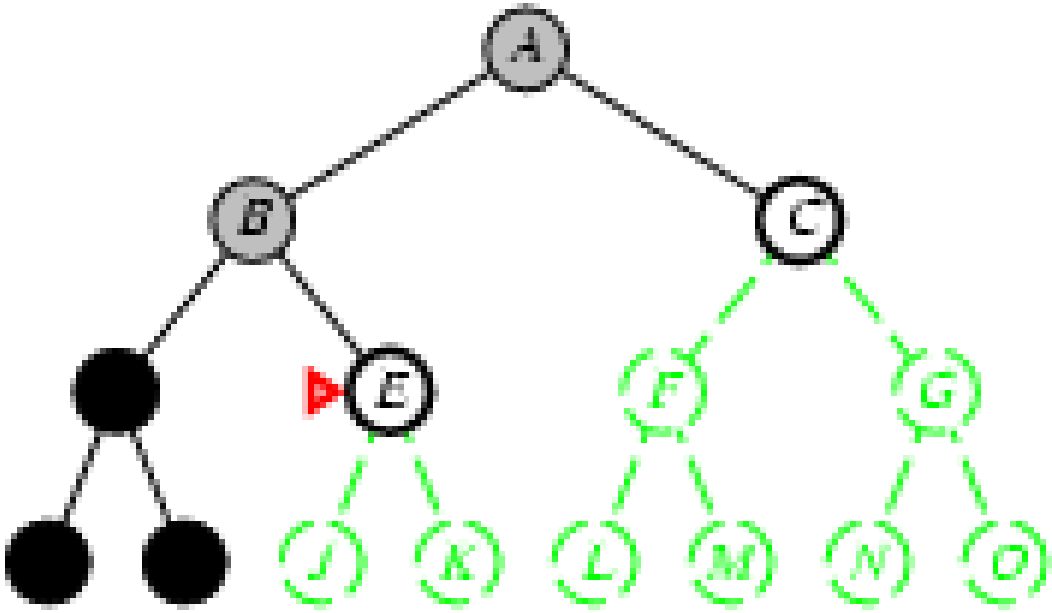
Búsqueda en profundidad

Ejemplo de exploración de nodos en búsqueda en profundidad



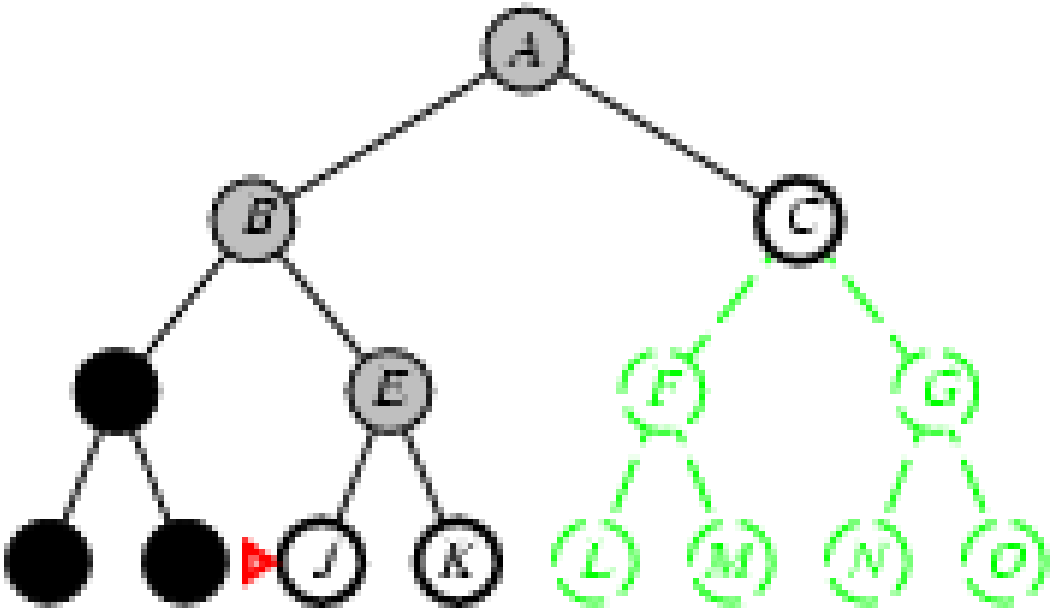
Búsqueda en profundidad

Ejemplo de exploración de nodos en búsqueda en profundidad



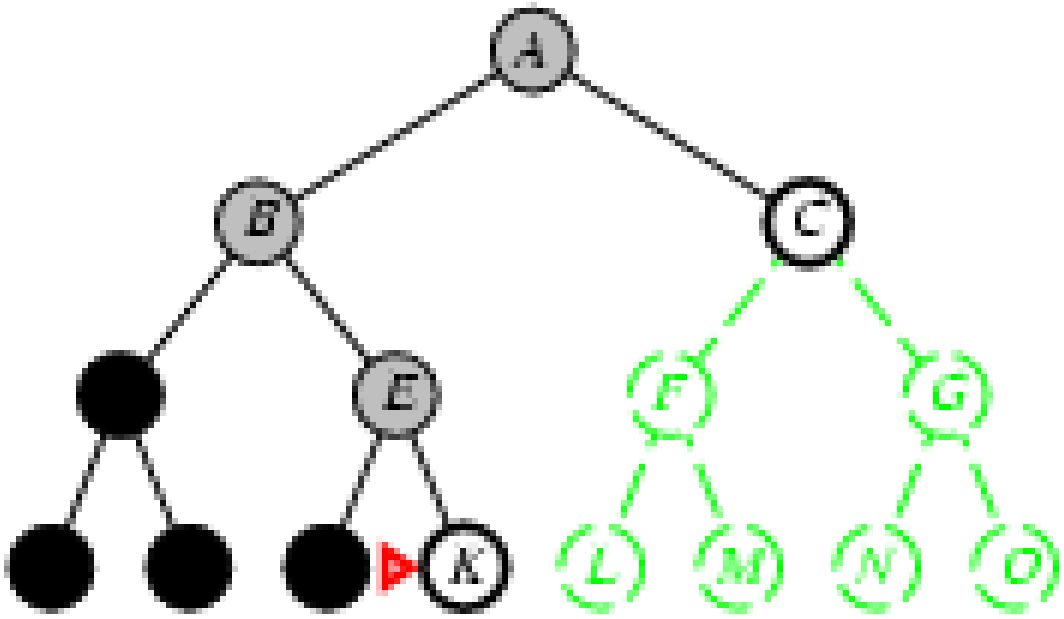
Búsqueda en profundidad

Ejemplo de exploración de nodos en búsqueda en profundidad



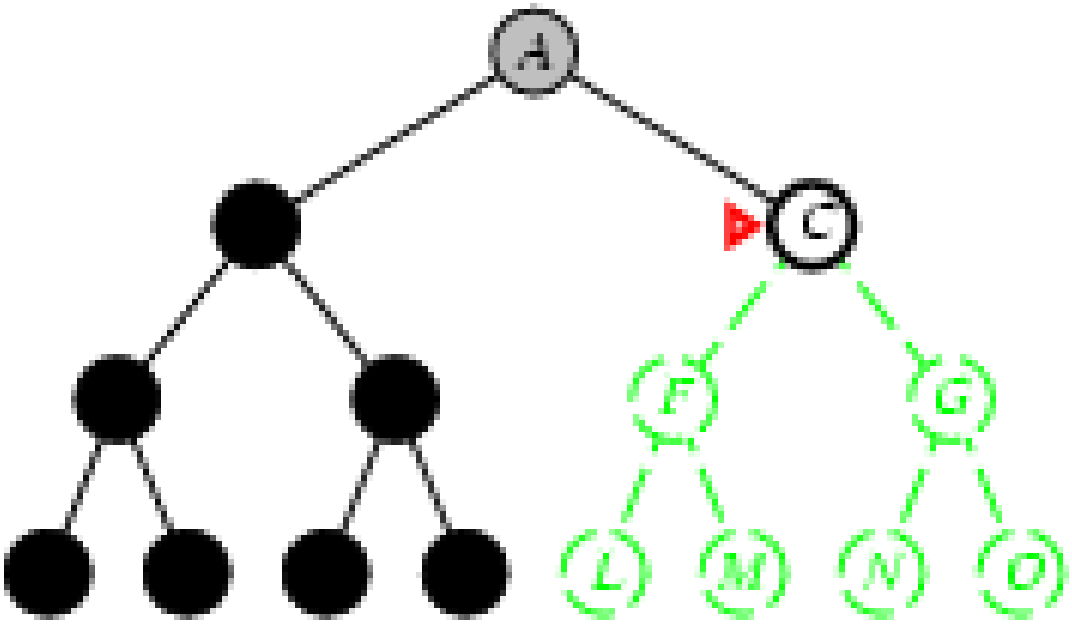
Búsqueda en profundidad

Ejemplo de exploración de nodos en búsqueda en profundidad



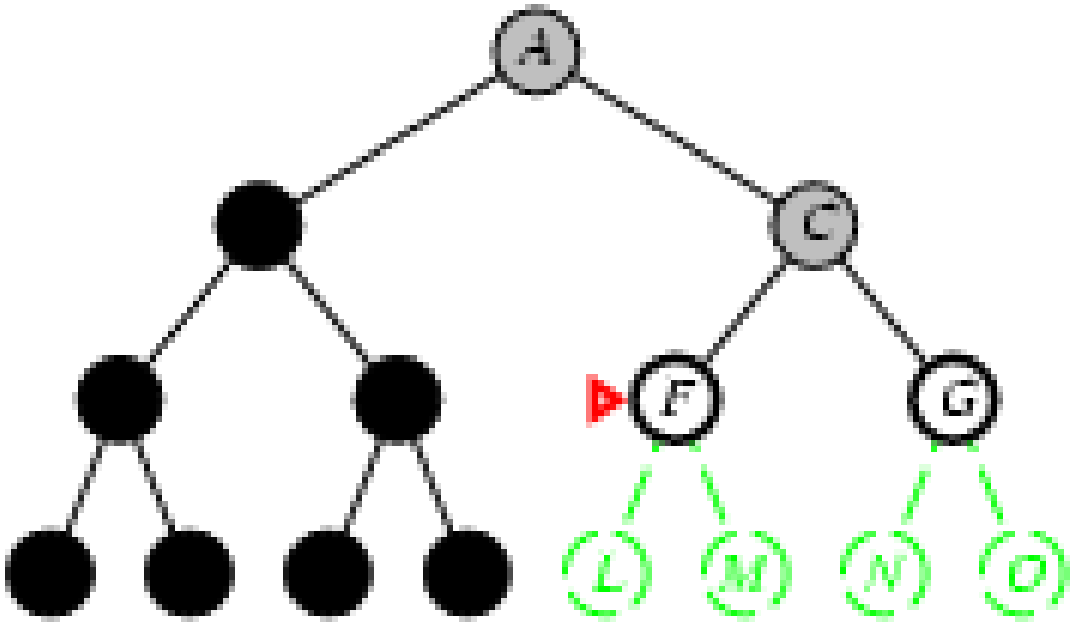
Búsqueda en profundidad

Ejemplo de exploración de nodos en búsqueda en profundidad



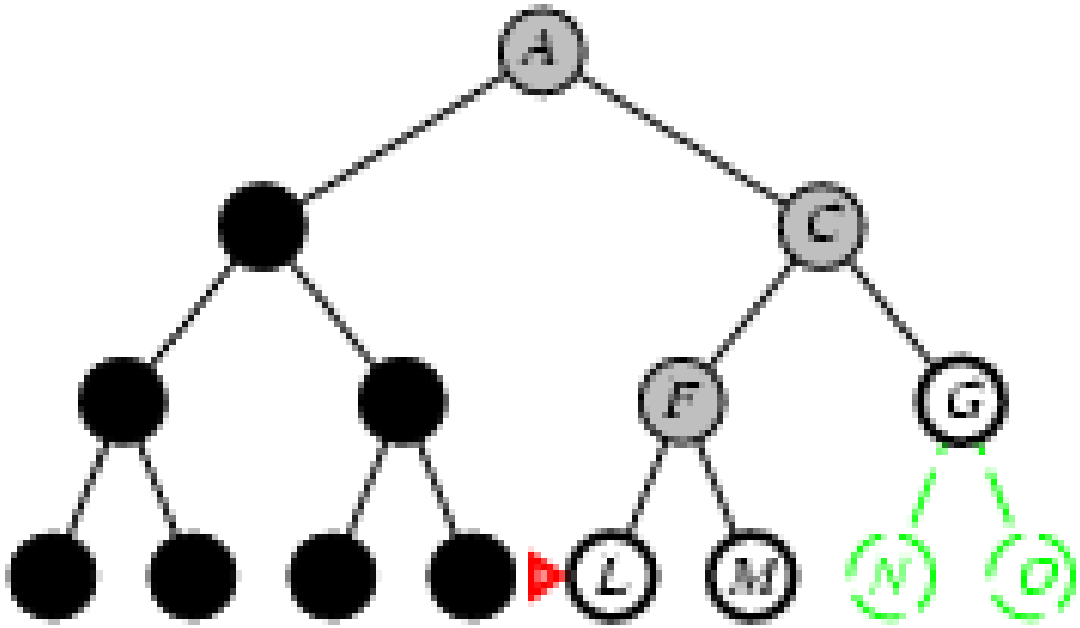
Búsqueda en profundidad

Ejemplo de exploración de nodos en búsqueda en profundidad



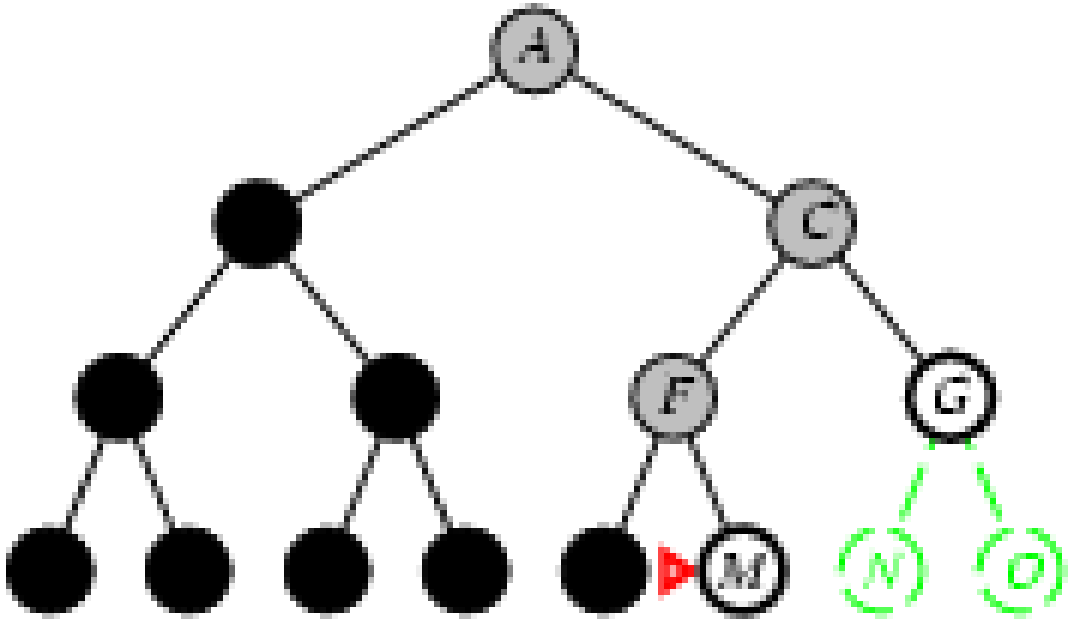
Búsqueda en profundidad

Ejemplo de exploración de nodos en búsqueda en profundidad



Búsqueda en profundidad

Ejemplo de exploración de nodos en búsqueda en profundidad



Búsqueda en profundidad

Propiedades de Búsqueda en Profundidad

- Completa? SI, solo en espacios con profundidad finita
- Complejidad de tiempo (Implementación TREE-SEARCH):

  $O(b^m)$, pésimo cuando m es mucho mayor que d, pero si hay muchas soluciones puede ser mas eficiente que la búsqueda en amplitud
- Complejidad de espacio (Implementación TREE-SEARCH):

  $O(bm)$, (complejidad lineal). En el ejemplo anterior con b=10, d=m=16 se tendría 156 kilobytes en lugar de 10 exabytes
- Optima? NO, ya que la búsqueda termina cuando encuentra la 1ra solución, pudiendo haber otra a una profundidad menor.

Búsqueda de costo uniforme

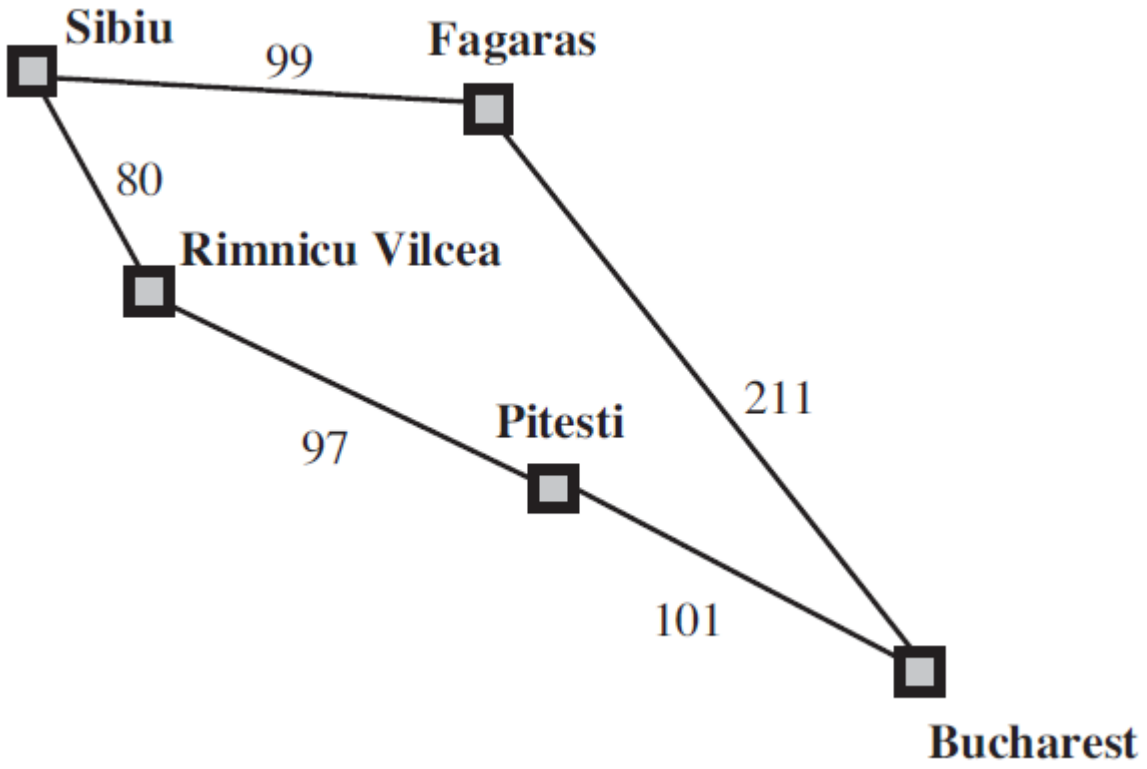
- Expande el nodo no expandido **n** que tenga el costo de camino **g(n)** más bajo
- Implementación: Puede ser GRAPH-SEARCH usando como frontera una lista ordenada por **g(n)** (PATH-COST)

```
function UNIFORM-COST-SEARCH(problem) returns a solution, or failure
  node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
  frontier ← a priority queue ordered by PATH-COST, with node as the only element
  explored ← an empty set
  loop do
    if EMPTY?(frontier) then return failure
    node ← POP(frontier)  /* chooses the lowest-cost node in frontier */
    if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
    add node.STATE to explored
    for each action in problem.ACTIONS(node.STATE) do
      child ← CHILD-NODE(problem, node, action)
      if child.STATE is not in explored or frontier then
        frontier ← INSERT(child, frontier)
      else if child.STATE is in frontier with higher PATH-COST then
        replace that frontier node with child
```

Búsqueda de costo uniforme

Búsqueda de costo uniforme (Ejercicio)

- Aplicar búsqueda de costo uniforme en el mapa de Rumania para llegar a *Bucharest* partiendo de *Sibiu*



Búsqueda de costo uniforme

Propiedades de Búsqueda de Costo Uniforme

- Equivalente a la búsqueda en amplitud si los costos de las acciones son todos iguales
- Completa? SI, si el costo de cada paso es ≥ ε
- Complejidad de tiempo:

  # de nodos con g() ≤ costo de solución óptima, $O(b^{1+\lceil 1+C^*/\varepsilon \rceil})$, *donde C* es el costo de la solución optima*
- Complejidad de espacio:

  Igual que la complejidad de tiempo
- Optima? SI, ya que los nodos son expandidos en orden creciente del costo total.

Búsqueda en Profundidad Limitada

- La búsqueda es hasta un limite de profundidad *l*. Para esto se considera que los nodos de profundidad *l* no tienen sucesores.
- Implementación recursiva:

```
function DEPTH-LIMITED-SEARCH(problem, limit) returns a solution, or failure/cutoff
  return RECURSIVE-DLS(MAKE-NODE(problem.INITIAL-STATE), problem, limit)

function RECURSIVE-DLS(node, problem, limit) returns a solution, or failure/cutoff
  if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
  else if limit = 0 then return cutoff
  else
      cutoff_occurred? ← false
      for each action in problem.ACTIONS(node.STATE) do
          child ← CHILD-NODE(problem, node, action)
          result ← RECURSIVE-DLS(child, problem, limit − 1)
          if result = cutoff then cutoff_occurred? ← true
          else if result ≠ failure then return result
      if cutoff_occurred? then return cutoff else return failure
```

Búsqueda en Profundidad Limitada

Propiedades de Búsqueda en Profundidad Limitada

- Completa? NO, la solución puede estar mas profunda que *l*
- Complejidad de tiempo: $O(b^l)$
- Complejidad de espacio: $O(bl)$,
- Optima? NO

Búsqueda con Profundización Iterativa

- Llama iterativamente a BFS limitado, aumentando gradualmente el limite de profundidad *l*

```
function ITERATIVE-DEEPENING-SEARCH(problem) returns a solution, or failure
    for depth = 0 to ∞ do
        result ← DEPTH-LIMITED-SEARCH(problem, depth)
        if result ≠ cutoff then return result
```

Búsqueda con Profundización Iterativa

Ejemplo de búsqueda en profundidad con profundización iterativa: *l=0*

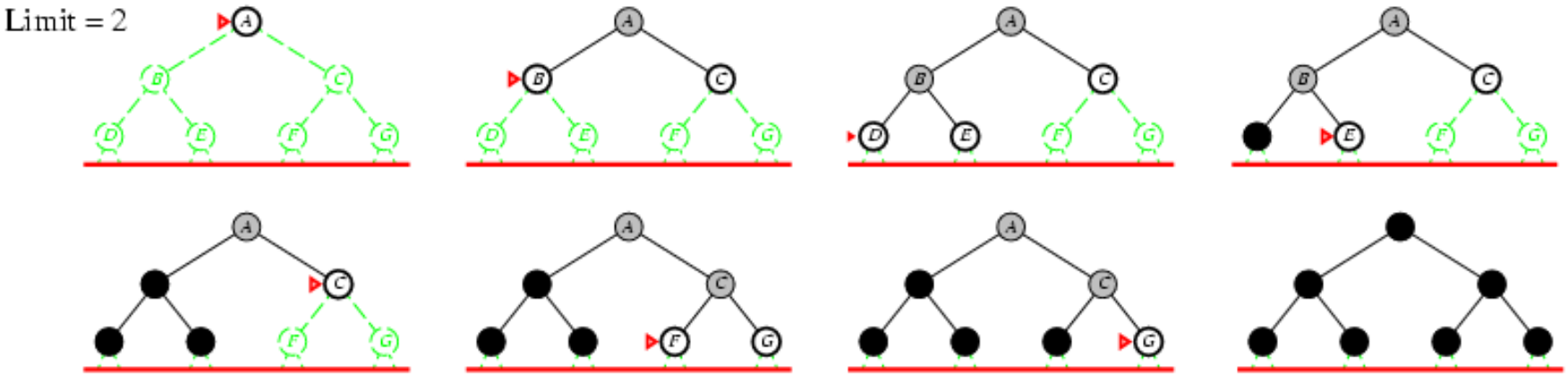
Limit = 0

Búsqueda con Profundización Iterativa

Ejemplo de búsqueda en profundidad con profundización iterativa: *l=1*

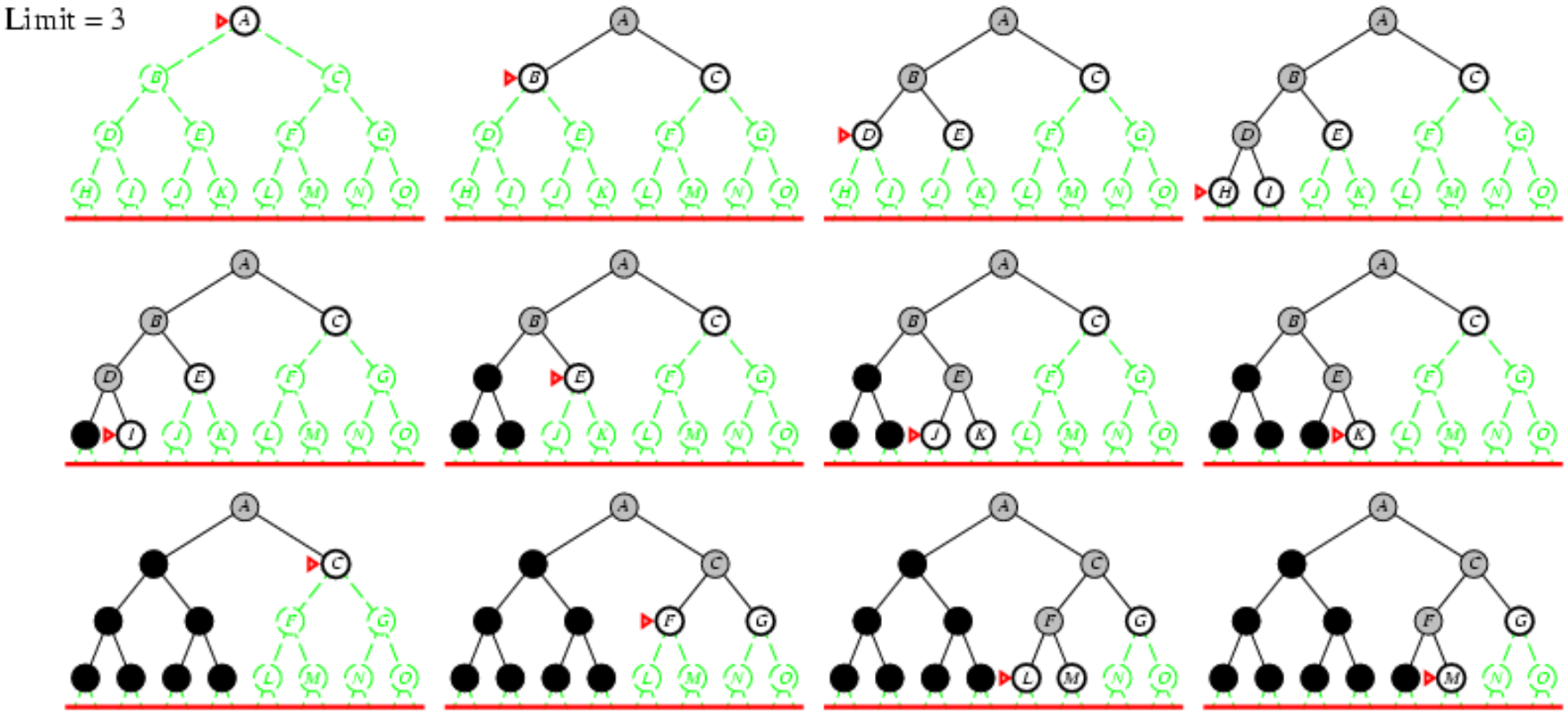
Búsqueda con Profundización Iterativa

Ejemplo de búsqueda en profundidad con profundización iterativa: *l=2*



Búsqueda con Profundización Iterativa

Ejemplo de búsqueda en profundidad con profundización iterativa: *l=3*



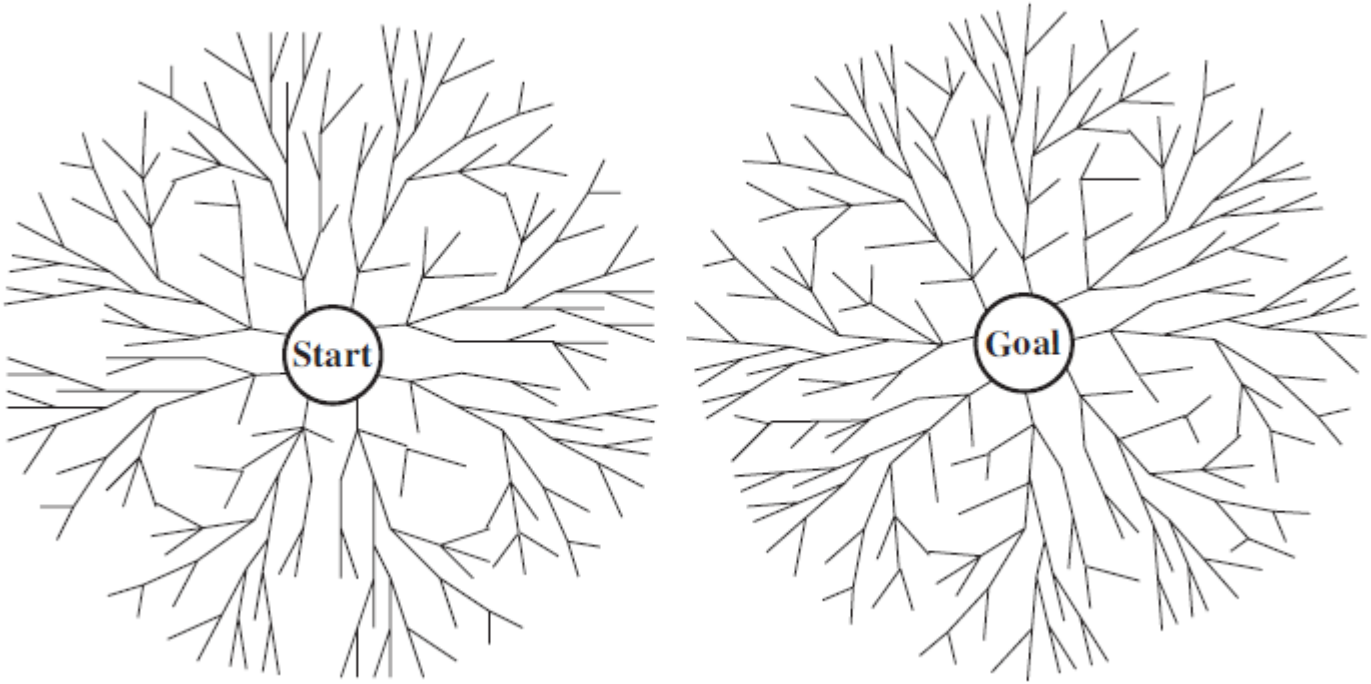
Búsqueda con Profundización Iterativa

Propiedades de búsqueda en profundidad con profundización iterativa

- Completa? SI, siempre encontrara un nivel donde este la solucion
- Complejidad de tiempo: $(d+1)b^0 + d\,b^1 + (d-1)b^2 + \ldots + b^d = O(b^d)$
- Complejidad de espacio: $O(bd)$,
- Optima? SI, si todas las acciones cuestan igual

Búsqueda Bidireccional

- Busca simultáneamente desde el estado inicial y desde el estado objetivo
- El test de objetivo chequea si las dos fronteras se intersectan



Búsqueda Bidireccional

Propiedades de búsqueda bidireccional

- Completa? SI, si ambas búsquedas son en amplitud
- Complejidad de tiempo: $O(b^{d/2}) + O(b^{d/2}) = O(b^{d/2})$
- Complejidad de espacio:
  - $O(b^{d/2}) + O(b^{d/2}) = O(b^{d/2})$, si ambas búsquedas son en amplitud
  - $O(b^{d/2}) + O(bd) = O(b^{d/2})$, si una búsqueda es en amplitud y la otra en profun.
- Optima? SI, si todas las acciones cuestan igual y si ambas busquedas son en amplitud

Resumen de estrategias de búsqueda

Resumen de estrategias de búsqueda sin información

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Complete?	Yes[a]	Yes[a,b]	Yes[a]	No	Yes[a]	Yes[a,d]
Time	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(b^m)$	$O(b^\ell)$	$O(b^d)$	$O(b^{d/2})$
Space	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(bm)$	$O(b\ell)$	$O(bd)$	$O(b^{d/2})$
Optimal?	Yes[c]	Yes	No	No	Yes[c]	Yes[c,d]

[a]: completo si b es finito

[b]: completo si costo de paso ≥ ε

[c]: optimo si todas las acciones cuestan igual

[d]: si ambas búsquedas son en amplitud

Preguntas?