Manara University
Faculty of Engineering
Department of IT

Introduction to Artificial Intelligence

Problem Solving and Search
Part I

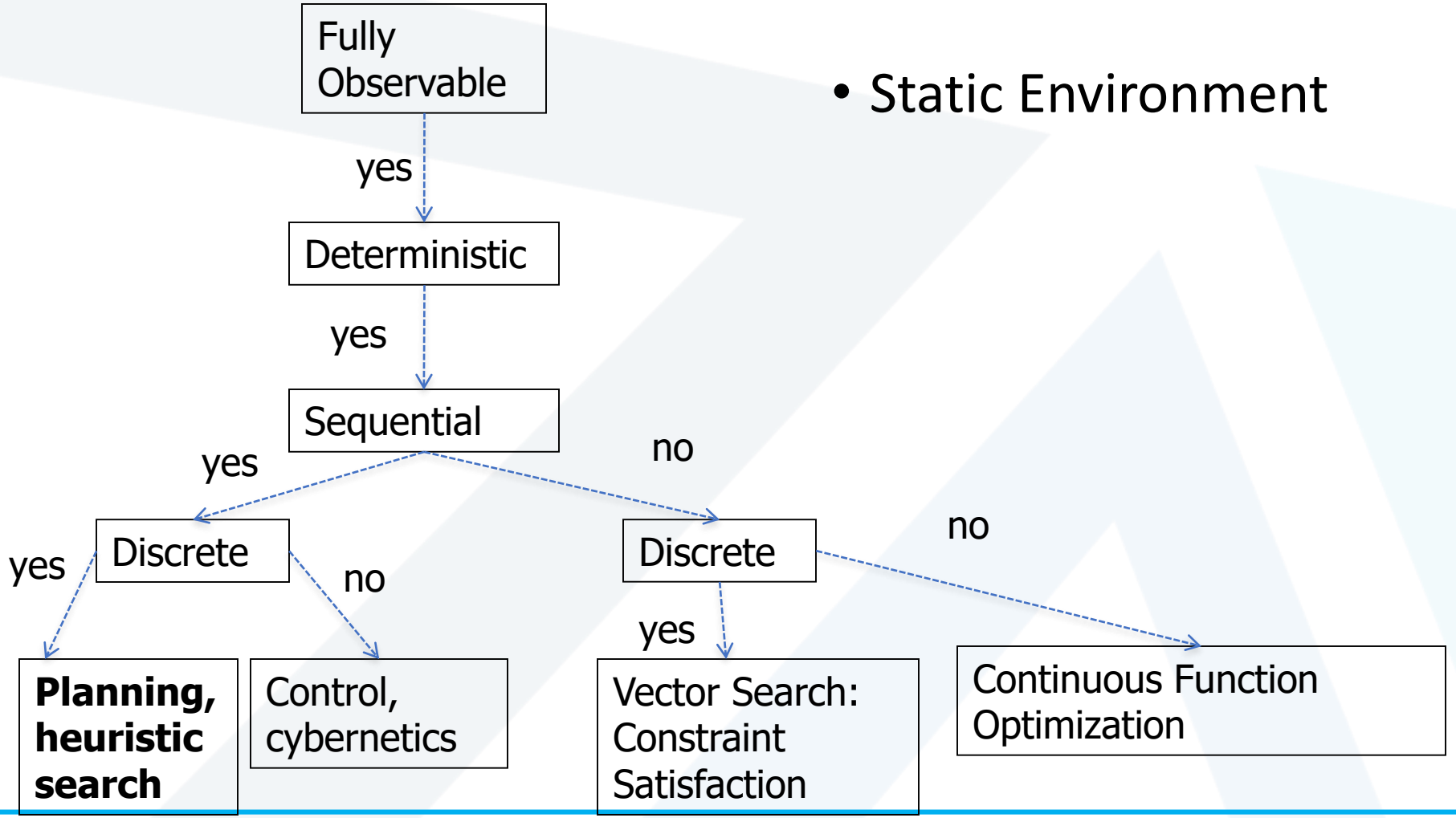
Lecture 3

Prof Dr. Eng. Mariam M. Saii

Outline

- Problem formulation: representing sequential problems.
- Example problems.
- Planning for solving sequential problems without uncertainty.
- Basic search algorithms
  - BFS
  - DFS
  - iterative

Environment Type Discussed In this Lecture



- Static Environment

الاستراتيجية العامة لحل المسائل

- التمثيل العام لحل المسائل
- تطبيق عمليات للانتقال من حالة الى اخرى
- السعي للوصول الى الهدف

مسألة المبشرين وأكلة لحوم البشر

- ثلاث مبشرين وثلاث من أكلة لحوم البشر على ضفة نهر (الضفة اليسرى) ويريدون الانتقال الى الضفة الأخرى (الضفة اليمنى للنهر)
- يوجد قارب يتسع لشخصين فقط
- يجب الا يزيد عدد أكلة لحوم البشر على عدد المبشرين في أي مرحلة من مراحل الانتقال

مسألة المبشرين وأكلة لحوم البشر

LEFT BANK الضفة اليسرى			RIGHT BANK الضفة اليمنى		
M	C	BOAT	M	C	BOAT
0	0	YES	3	3	NO
0	1	YES	3	2	NO
0	2	YES	3	1	NO
0	3	YES	3	0	NO
1	0	YES	2	3*	NO
1	1	YES	2	2	NO
1	2*	YES	2	1	NO
1	3*	YES	2	0	NO
2	0	YES	1	3*	NO
2	1	YES	1	2*	NO
2	2	YES	1	1	NO
2	3*	YES	1	0	NO
3	0	YES	0	3	NO
3	1	YES	0	2	NO
3	2	YES	0	1	NO
3	3	YES	0	0	NO
0	0	NO	3	3	YES
0	1	NO	3	2	YES
0	2	NO	3	1	YES
0	3	NO	3	0	YES
1	0	NO	2	3*	YES
1	1	NO	2	2	YES
1	2*	NO	2	1	YES
1	3*	NO	2	0	YES
2	0	NO	1	3*	YES
2	1	NO	1	2*	YES
2	2	NO	1	1	YES
2	3*	NO	1	0	YES
3	0	NO	0	3	YES
3	1	NO	0	2	YES
3	2	NO	0	1	YES
3	3	NO	0	0	YES

- * تدل على أن الحالة غير جائزة وهي زيادة عدد أكلة لحوم البشر على عدد المبشرين

مسألة المبشرين وأكلة لحوم البشر

الضفة اليسرى LEFT BANK			الضفة اليمنى RIGHT BANK		
M	C	BOAT	M	C	BOAT
0	0	YES*	3	3	NO
0	1	YES	3	2	NO
0	2	YES	3	1	NO
0	3	YES	3	0	NO
1	1	YES	2	2	NO
2	2	YES	1	1	NO
3	0	YES*	0	3	NO
3	1	YES	0	2	NO
3	2	YES	0	1	NO
3	3	YES	0	0	NO
0	0	NO	3	3	YES
0	1	NO	3	2	YES
0	2	NO	3	1	YES
0	3	NO*	3	0	YES
1	1	NO	2	2	YES
2	2	NO	1	1	YES
3	0	NO	0	3	YES
3	1	NO	0	2	YES
3	2	NO	0	1	YES
3	3	NO*	0	0	YES

- * تدل على أن الحالة تتنافى مع الفرض (غير جائزة) وهي زيادة عدد أكلة لحوم البشر على عدد المبشرين

مسألة المبشرين وأكلة لحوم البشر – العمليات الممكنة

العملية	العمليات الممكنة
O1	انقل مبشراً من الضفة اليسرى إلى اليمنى
O2	انقل متوحشاً من الضفة اليسرى إلى اليمنى
O3	انقل مبشراً ومتوحشاً من الضفة اليسرى إلى اليمنى
O4	انقل مبشرين من الضفة اليسرى إلى اليمنى
O5	انقل متوحشين من الضفة اليسرى إلى اليمنى
O6	انقل مبشراً من الضفة اليمنى إلى اليسرى
O7	انقل متوحشاً من الضفة اليمنى إلى اليسرى
O8	انقل مبشراً ومتوحشاً من الضفة اليمنى إلى اليسرى
O9	انقل مبشرين من الضفة اليمنى إلى اليسرى
O10	انقل متوحشين من الضفة اليمنى إلى اليسرى

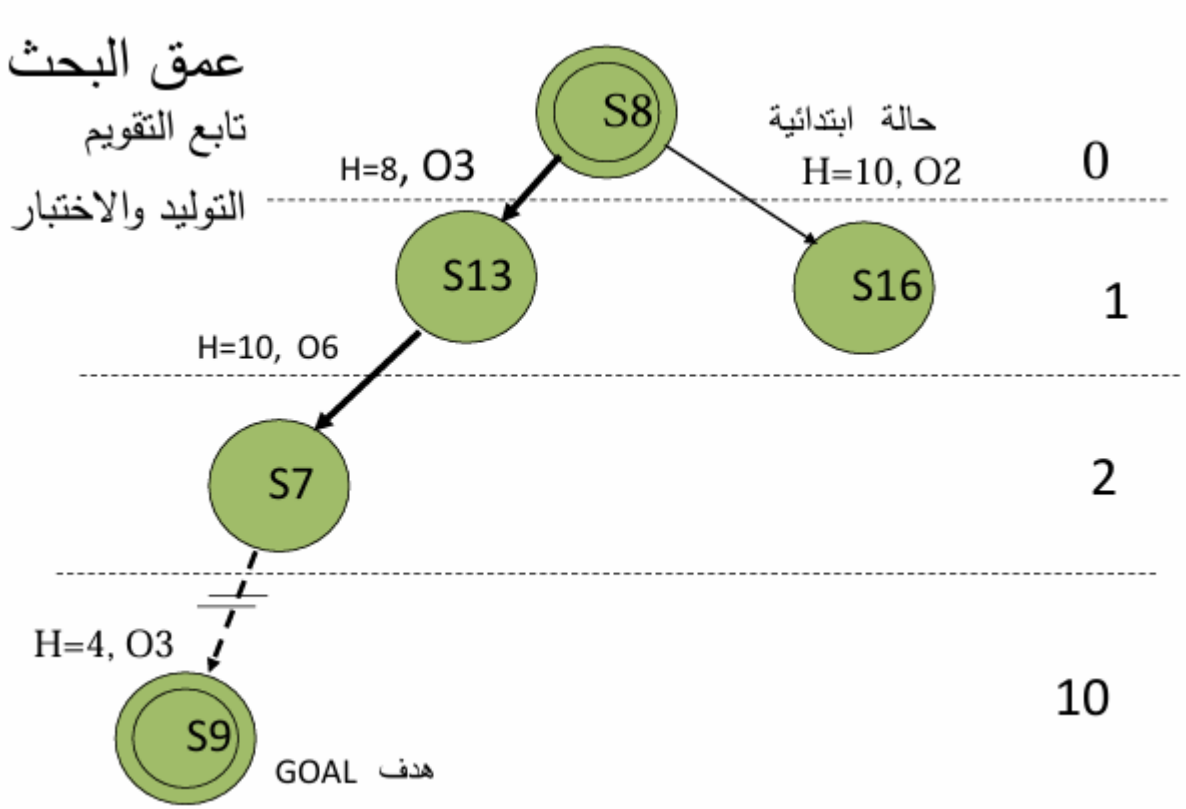
مسألة المبشرين وأكلة لحوم البش- الحالات الممكنة

الحالة	الضفة اليسرى LEFT BANK			الضفة اليمنى RIGHT BANK		
الموجودة	M	C	BOAT	M	C	BOAT
S1	0	1	YES	3	2	NO
S2	0	2	YES	3	1	NO
S3	0	3	YES	3	0	NO
S4	1	1	YES	2	2	NO
S5	2	2	YES	1	1	NO
S6	3	1	YES	0	2	NO
S7	3	2	YES	0	1	NO
S8	3	3	YES	0	0	NO
S9	0	0	NO	3	3	YES
S10	0	1	NO	3	2	YES
S11	0	2	NO	3	1	YES
S12	1	1	NO	2	2	YES
S13	2	2	NO	1	1	YES
S14	3	0	NO	0	3	YES
S15	3	1	NO	0	2	YES
S16	3	2	NO	0	1	YES

مسألة المبشرين وأكلة لحوم البشر- الانتقالات

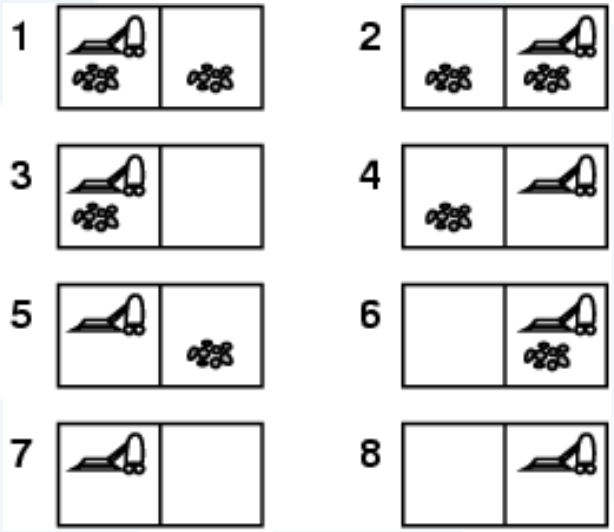
الحالة الموجودة		الضفة اليسرى LEFT BANK			الضفة اليمنى RIGHT BANK		
		M	C	BOAT	M	C	BOAT
O3	S8	3	3	YES	0	0	NO
O6	S12	2	2	NO	1	1	YES
O5	S7	3	2	YES	0	1	NO
O7	S6	3	0	NO	0	3	YES
O4	S12	3	1	YES	0	2	NO
O8	S5	1	1	NO	2	2	YES
O4	S11	2	2	YES	1	1	NO
O7	S3	0	2	NO	3	1	YES
O5	S10	0	3	YES	3	0	NO
O3	S4	0	1	NO	3	2	YES
G	S9	0	0	NO	3	3	YES

مسألة المبشرين وأكلة لحوم البشر – مخطط الحالة



Sequential Action Example

- Deterministic, fully observable → single-state problem
  - Agent knows exactly which state it will be in; solution is a sequence
  - Vacuum world → everything observed
  - Romania → The full map is observed

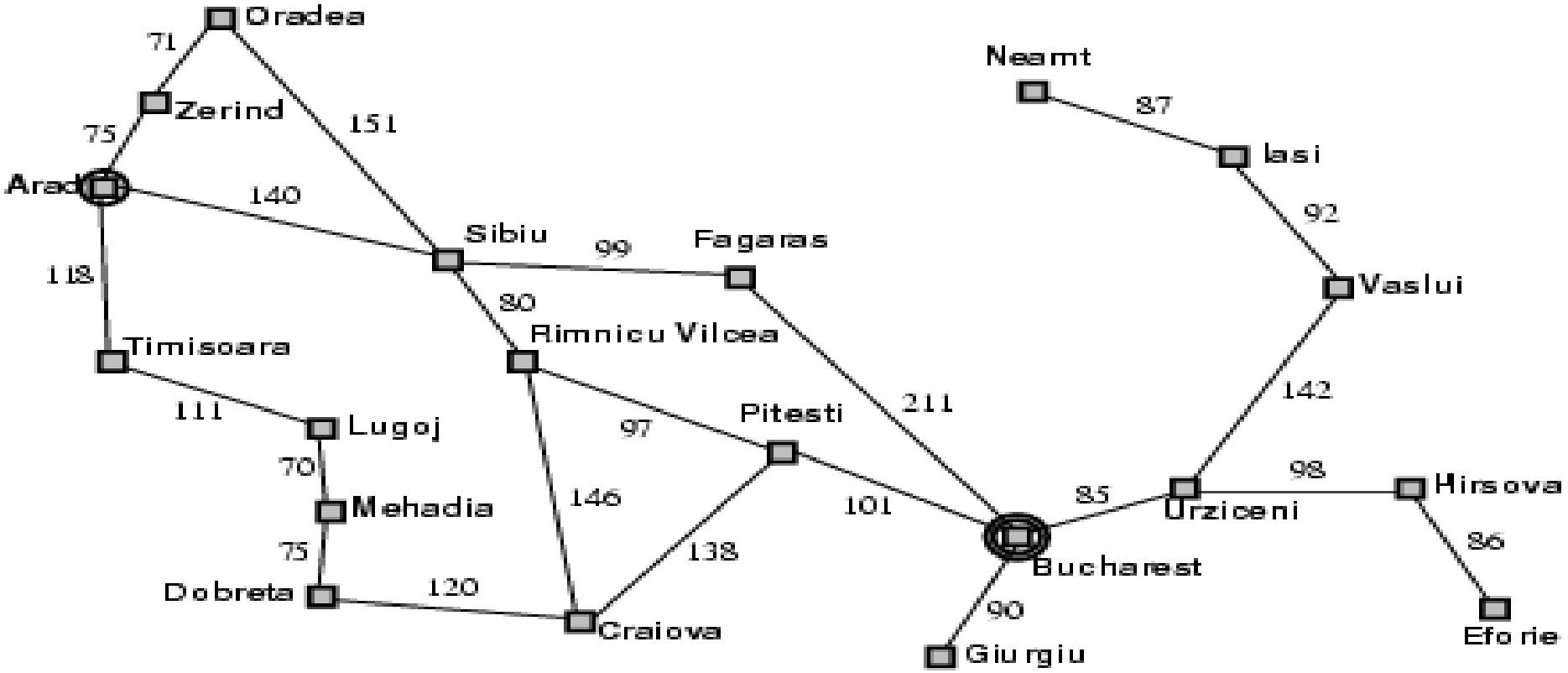


- Single-state: Start in #5. Solution??
  - [Right, Suck]

Example: Romania

- On holiday in Romania; currently in Arad.
- Flight leaves tomorrow from Bucharest
-
- Formulate goal:
  - be in Bucharest
  -
- Formulate problem:
  - states: various cities
  - actions: drive between cities
- Find solution:
  - sequence of cities, e.g., Arad, Sibiu, Fagaras, Bucharest
  -

Example: Romania



Abstraction: The process of removing details from a representation
Is the map a good representation of the problem? What is a good replacement?

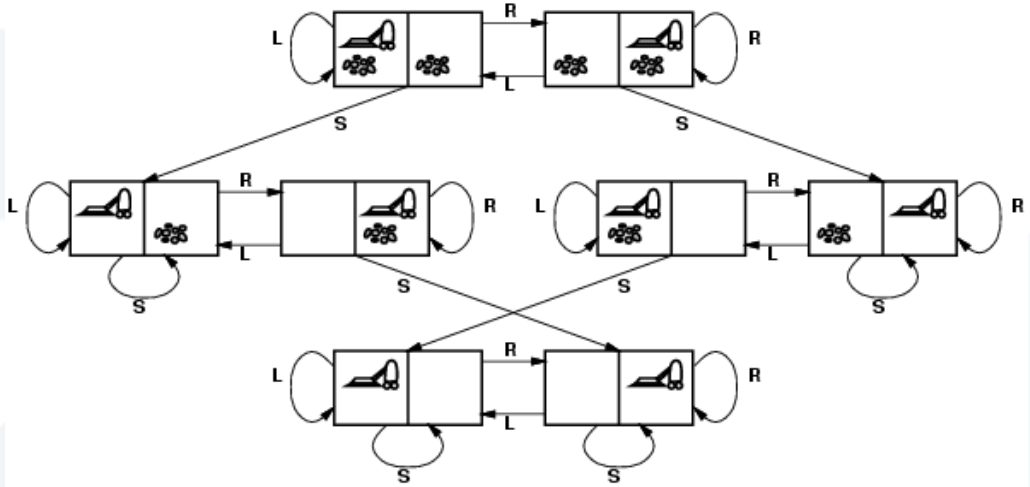
Single-state problem formulation

A problem is defined by four items:

1. initial state e.g., "at Arad“
2. actions or successor function *S(x)* = set of action–state pairs
   - e.g., *S(Arad)* = {<*Arad → Zerind, Zerind*>, … }
3. goal test, can be
   - explicit, e.g., *x* = "at Bucharest"
   - implicit, e.g., *Checkmate(x)*
4. path cost (additive)
   - e.g., sum of distances, number of actions executed, etc.
   - *c(x,a,y)* is the step cost, assumed to be ≥ 0

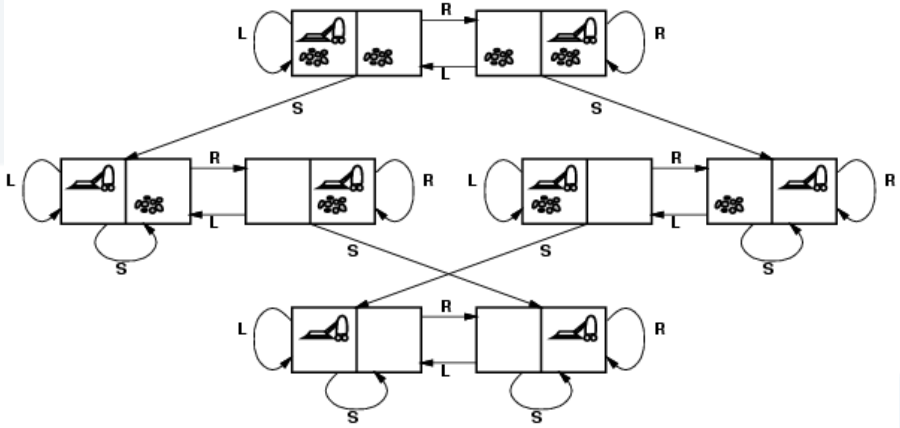
- A solution is
  - a sequence of actions leading from the initial state to a goal state

Vacuum world state space graph



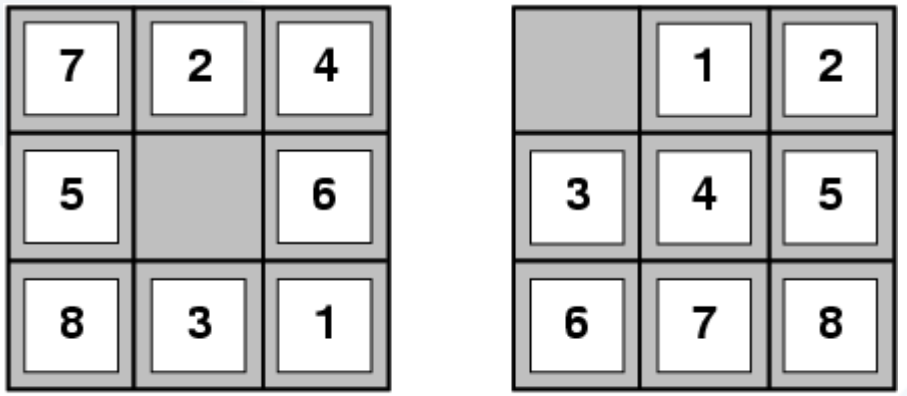
- states?
- actions?
- goal test?
- path cost?
-

Vacuum world state space graph



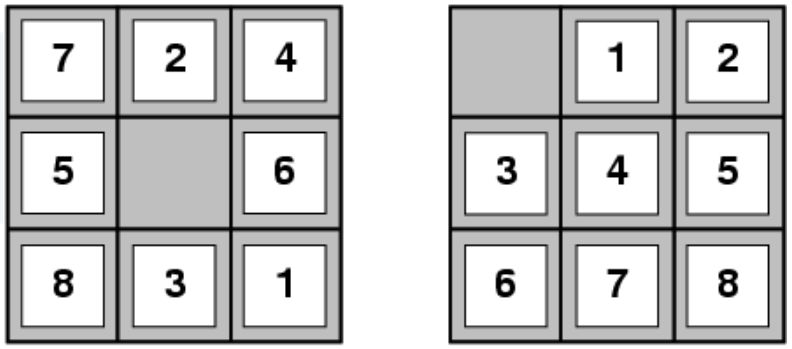
- states? integer dirt and robot location
- actions? *Left*, *Right*, *Suck*
- goal test? no dirt at all locations
- path cost? 1 per action

Example: The 8-puzzle



- states?
- actions?
- goal test?
- path cost?

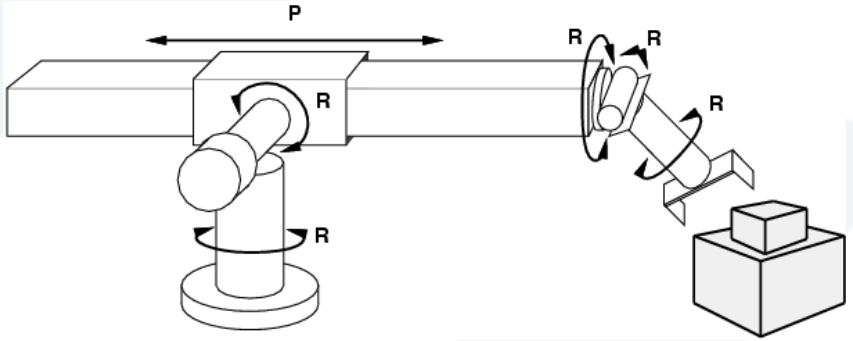
Example: The 8-puzzle



- states? locations of tiles
- actions? move blank left, right, up, down
- goal test? = goal state (given)
- path cost? 1 per move

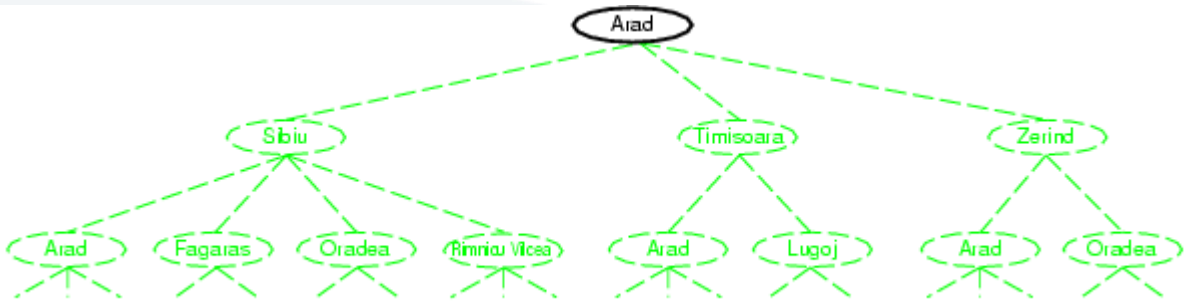
[Note: optimal solution of *n*-Puzzle family is NP-hard]

Example: robotic assembly

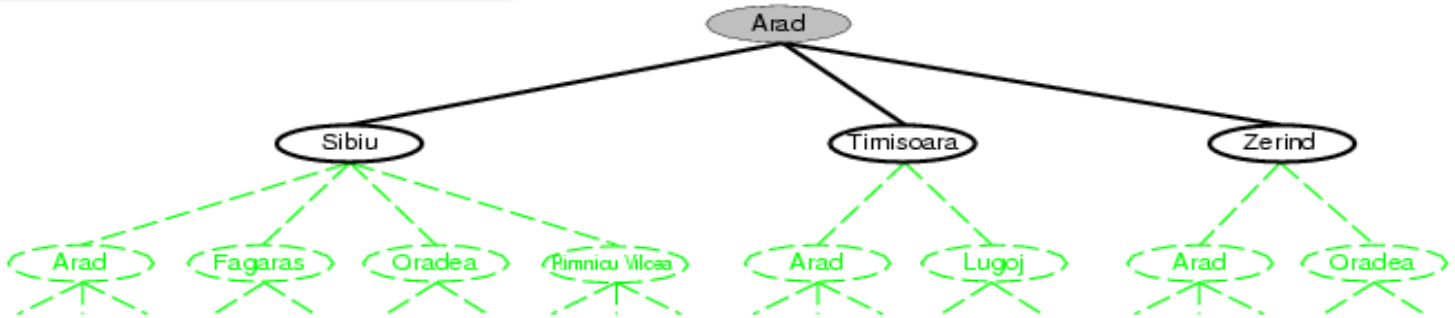


- states?:
  - real-valued coordinates of robot joint angles parts of the object to be assembled
  -
- actions?:
  - continuous motions of robot joints
- goal test?:
  - complete assembly
  -
- path cost?:
  - time to execute
  -

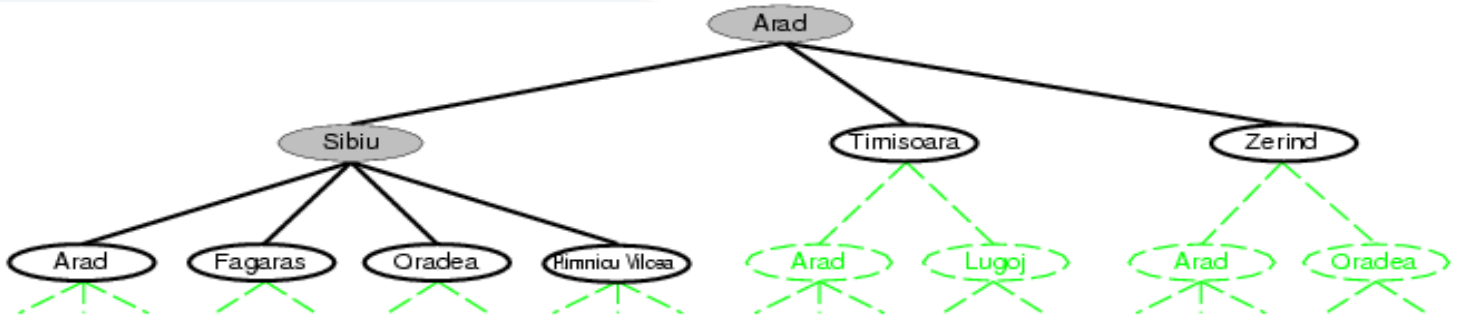
Tree search example



Tree search example



Tree search example



Search Graph vs. State Graph

- Be careful to distinguish
  - Search tree: nodes are **sequences of actions.**
  - State Graph: Nodes are states of the environment.
  - We will also consider soon **search graphs**.
- Demo: http://aispace.org/search/

Search strategies معايير تقويم البحث

- Strategies are evaluated along the following dimensions:
  - الشمولية completeness: does it always find a solution if one exists?
    - تعبر عما اذا كانت الاستراتيجية تضمن الوصول الى الحل في حال وجوده
  - time complexity: number of nodes generated
    - كم نستغرق للوصول الى الحل ويرتبط بعدد العقد الموسعة
  - space complexity: maximum number of nodes in memory
    - يعبر عن حجم الذاكرة المستهلكة في الحالة الأسوأ ويرتبط بالعجدد الأعظمي للعقد المخزنة في الذاكرة
  - optimality: does it always find a least-cost solution?
    - هل نصل دائما الى الحل ذو الكلفة الأقل (العمق والممر الأفضل)

Time and space complexity

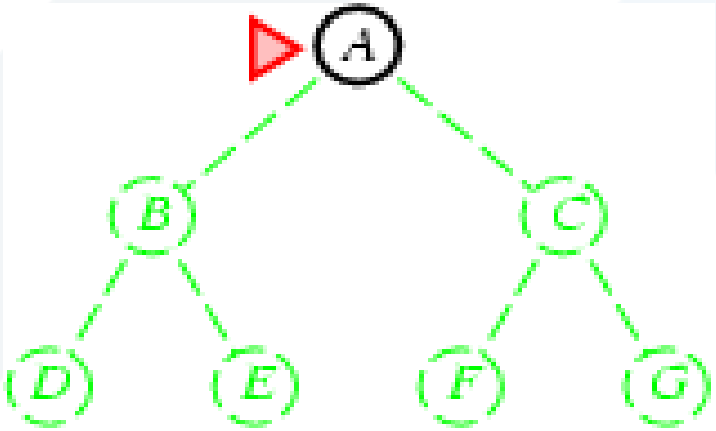
- Time and space complexity are measured in terms of
  - *b:* maximum branching factor of the search معامل التفريع الأعظمي لشجرة البحث (عدد العقد الأبناء الأعظمي التي تم توليدها)
  - *d:* depth of the least-cost solution عمق الحل الأقل تكلفة
  - *m*: maximum depth of the state space (may be ∞) العمق الأعظمي لفضاء الحالة (قد يكون لا نهائي)

Search strategies

- Uninformed البحث الأعمى search strategies use only the information available in the problem definition
  - Breadth-first search
  - Depth-first search
  - Depth-limited search
  - Iterative deepening search
- informed البحث المعرفي search strategies use knowledge about the problem definition
  - Best- first search
  - A* search

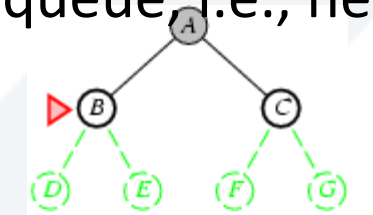
Breadth-first search

- Expand shallowest unexpanded node
-
- Implementation:
  - *Frontier* or *fringe* is a FIFO queue, i.e., new successors go at end
  -



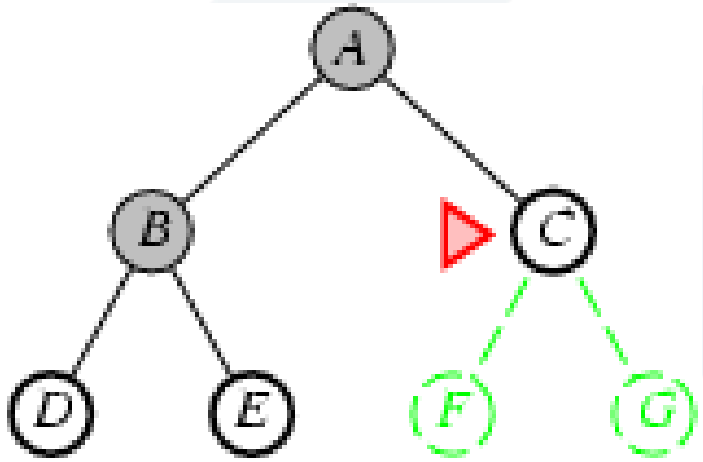
Breadth-first search

- Expand shallowest unexpanded node
-
- Implementation:
  - *frontier* is a FIFO queue, i.e., new successors go at end
  -



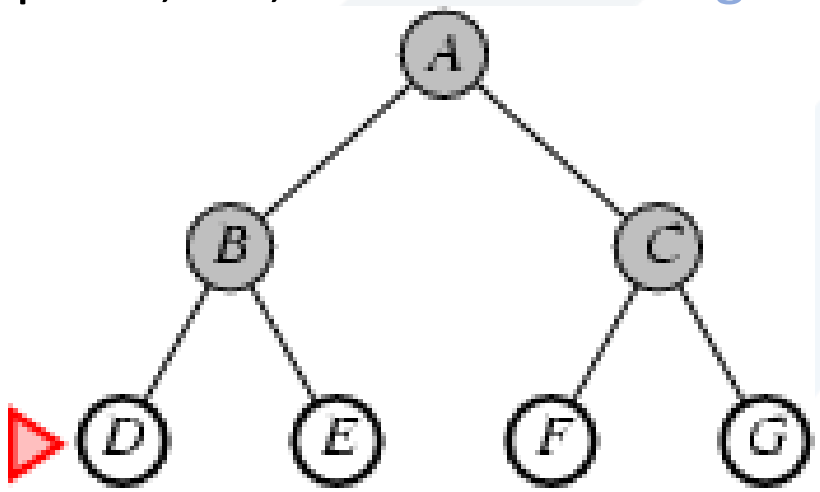
Breadth-first search

- Expand shallowest unexpanded node
-
- Implementation:
  - *frontier* is a FIFO queue, i.e., new successors go at end
  -



Breadth-first search

- Expand shallowest unexpanded node
- http://aispace.org/search/
- Implementation:
  - *frontier* is a FIFO queue, i.e., new successors go at end
  -



Properties of breadth-first search

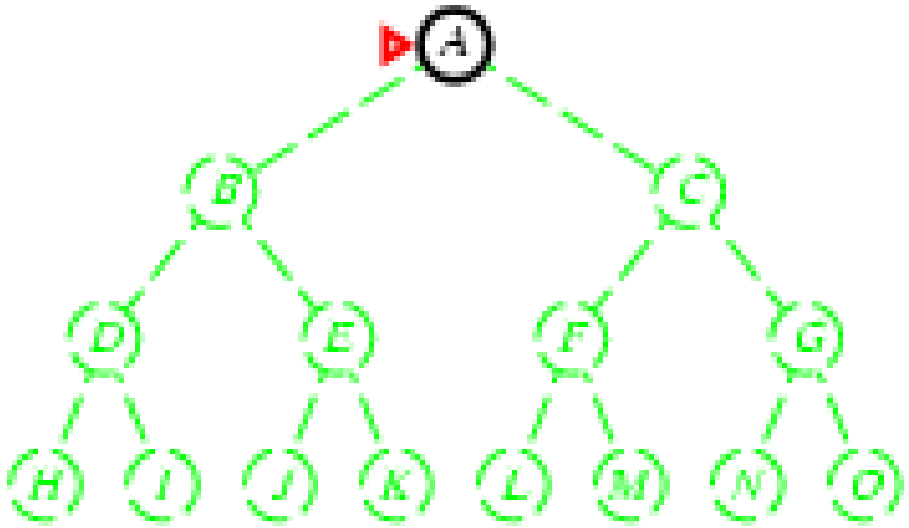
- Complete? Time? Space?Optimal?

- Complete? Yes (if b is finite)
-
- Time? $1+b+b^2+b^3+\ldots+b^d+b(b^d-1) = O(b^{d+1})$
-
- Space? $O(b^{d+1})$ (keeps every node in memory)
-
- Optimal? Yes (if cost = 1 per step)
-

- Space is the bigger problem (more than time)
-

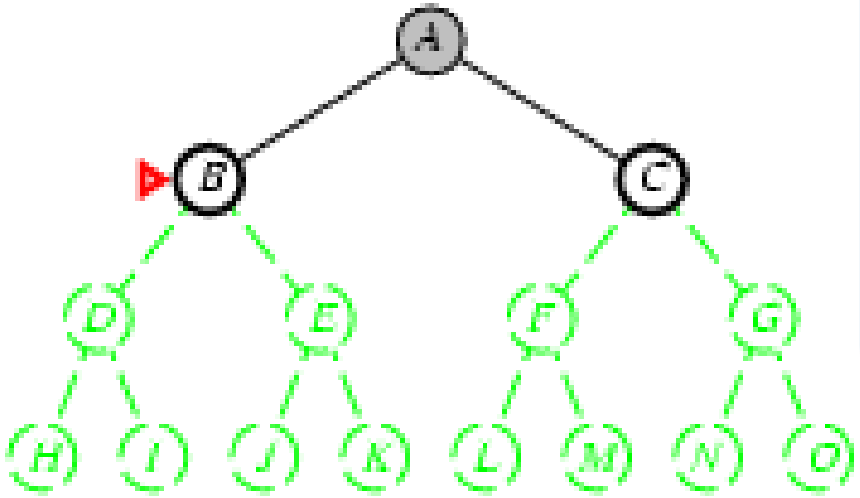
Depth-first search

- Expand deepest unexpanded node
-
- Implementation:
  - *frontier* = LIFO queue, i.e., put successors at front
  -



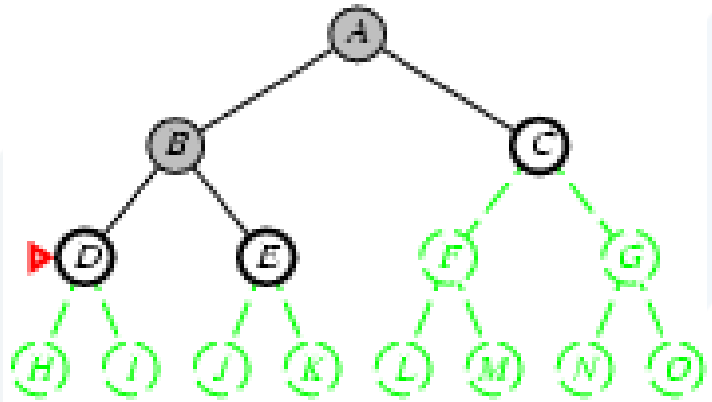
Depth-first search

- Expand deepest unexpanded node
-
- Implementation:
  - *frontier* = LIFO queue, i.e., put successors at front
  -



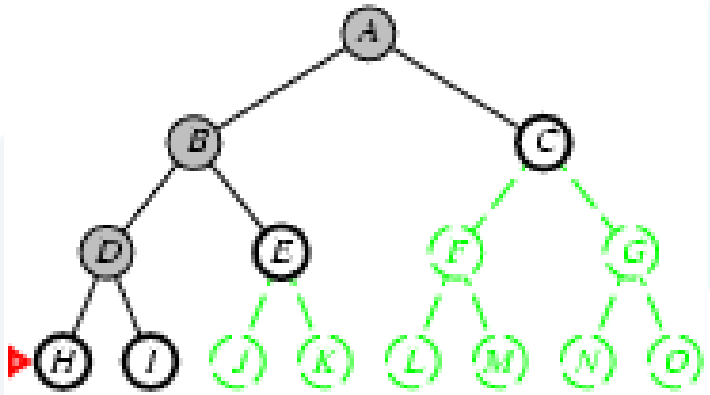
Depth-first search

- Expand deepest unexpanded node
-
- Implementation:
  - *frontier* = LIFO queue, i.e., put successors at front
  -



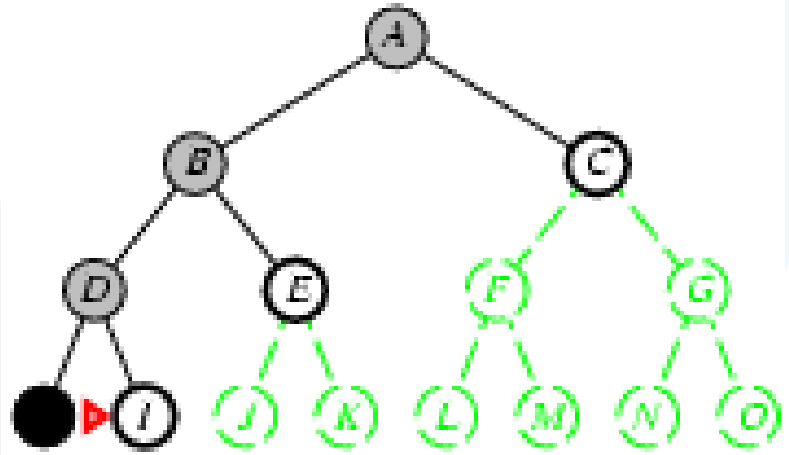
Depth-first search

- Expand deepest unexpanded node
-
- Implementation:
  - *frontier* = LIFO queue, i.e., put successors at front
  -



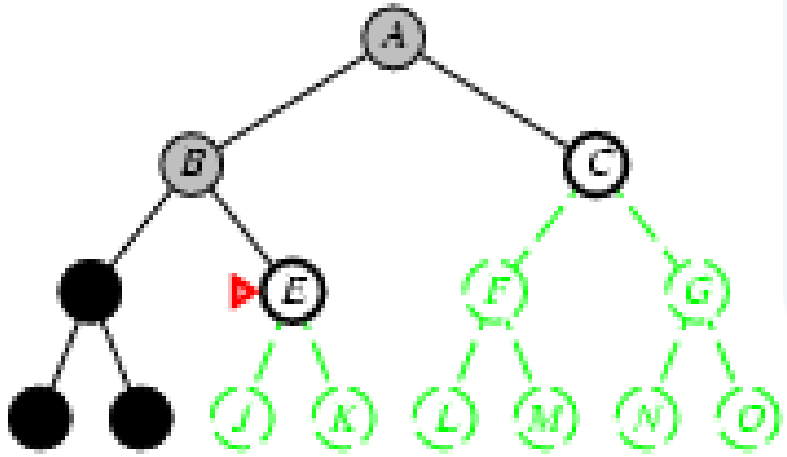
Depth-first search

- Expand deepest unexpanded node
-
- Implementation:
  - *frontier* = LIFO queue, i.e., put successors at front
  -



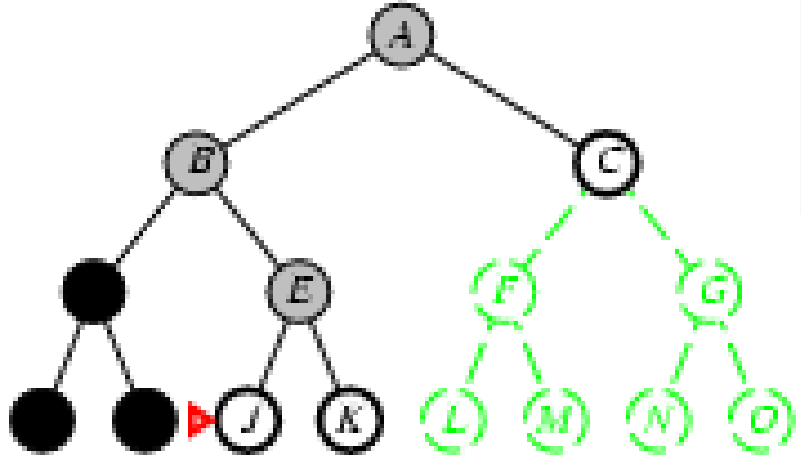
Depth-first search

- Expand deepest unexpanded node
-
- Implementation:
  - *frontier* = LIFO queue, i.e., put successors at front
  -



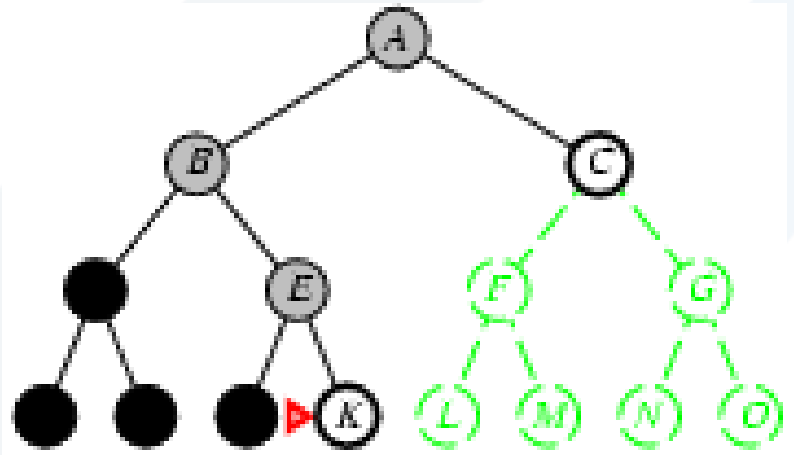
Depth-first search

- Expand deepest unexpanded node
-
- Implementation:
  - *frontier* = LIFO queue, i.e., put successors at front
  -



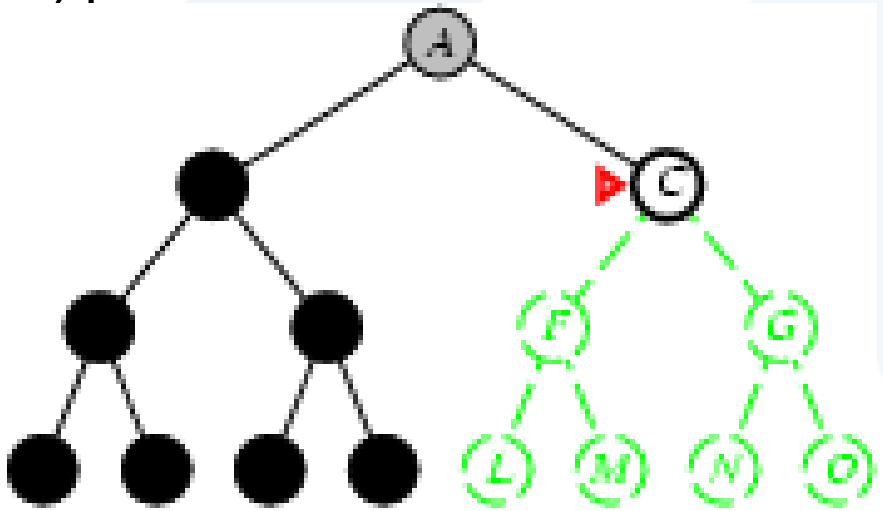
Depth-first search

- Expand deepest unexpanded node
-
- Implementation:
  - *frontier* = LIFO queue, i.e., put successors at front
  -



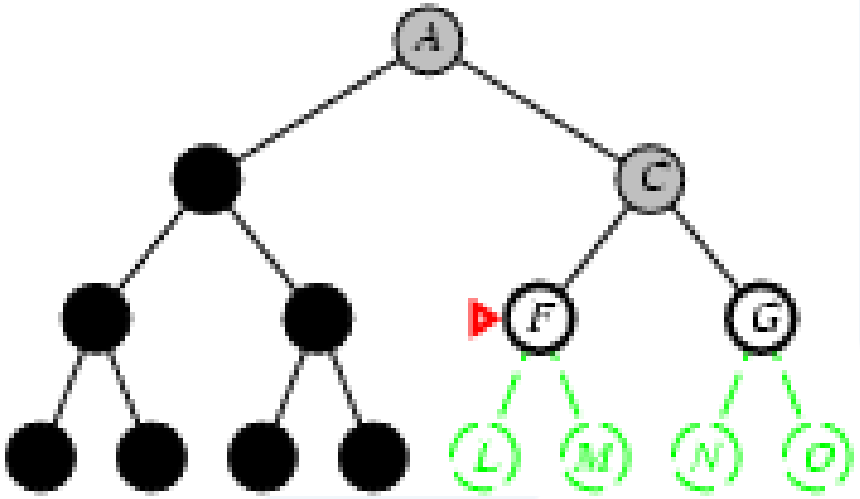
Depth-first search

- Expand deepest unexpanded node
-
- Implementation:
  - *frontier* = LIFO queue, i.e., put successors at front
  -



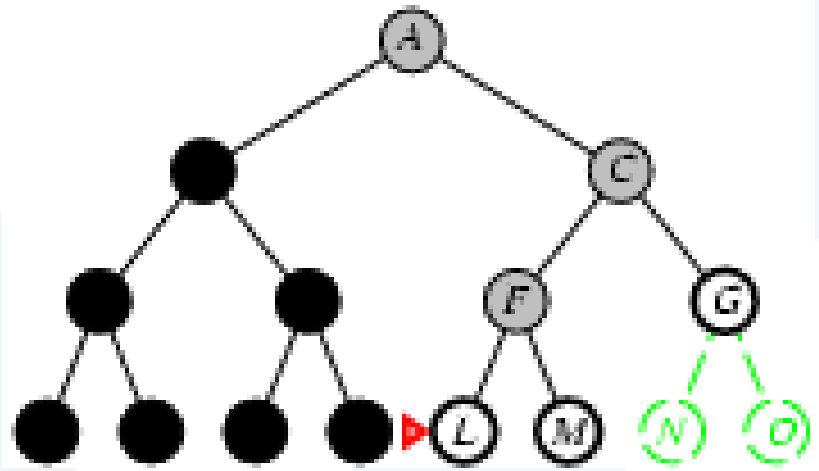
Depth-first search

- Expand deepest unexpanded node
-
- Implementation:
  - *frontier* = LIFO queue, i.e., put successors at front
  -



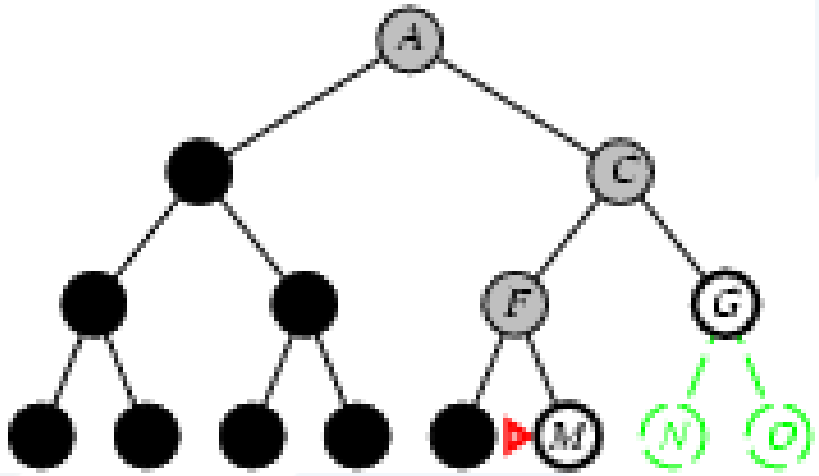
Depth-first search

- Expand deepest unexpanded node
-
- Implementation:
  - *frontier* = LIFO queue, i.e., put successors at front
  -



Depth-first search

- Expand deepest unexpanded node
- http://aispace.org/search/
- Implementation:
  - *frontier* = LIFO queue, i.e., put successors at front
  -



Properties of depth-first search

- Complete? Time? Space?Optimal?

- Complete? No: fails in infinite-depth spaces, spaces with loops
  - Modify to avoid repeated states along path (graph search)
  - → complete in finite spaces

- Time? $O(b^m)$: terrible if maximum depth m is much larger than solution depth d
  - but if solutions are dense, may be much faster than breadth-first

- Space? $O(bm)$, i.e., linear space! Store single path with unexpanded siblings.
  - Seems to be common in animals and humans.

- Optimal? No.

- Important for exploration (on-line search).

Depth-limited search

- depth-first search with depth limit *l*,
  - i.e., nodes at depth *l* have no successors
  - Solves infinite loop problem
- Common AI strategy: let user choose search/resource bound.
- Complete? No if l < d:
- Time? $O(b^l)$:
- Space? $O(bl)$, i.e., linear space!
- Optimal? No if l > b

Iterative deepening search

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

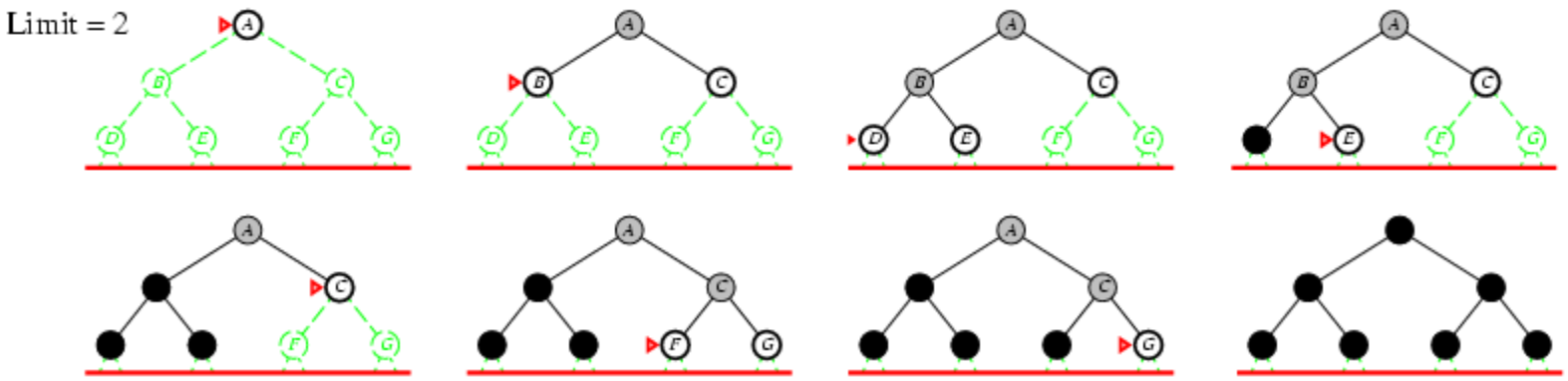
Iterative deepening search /=0

Limit = 0

Iterative deepening search /=1

Limit = 1

Iterative deepening search /=2



Iterative deepening search /=3