# Adapted from AIMA

# Artificial Intelligence
# Uninformed search

# Outline

- The "symbols&search" hypothesis for AI
- Problem-solving agents
  - A kind of goal-based agent
- Problem types
  - Single state (fully observable)
  - Search with partial information
- Problem formulation
  - Example problems
- Basic search algorithms
  - Uninformed

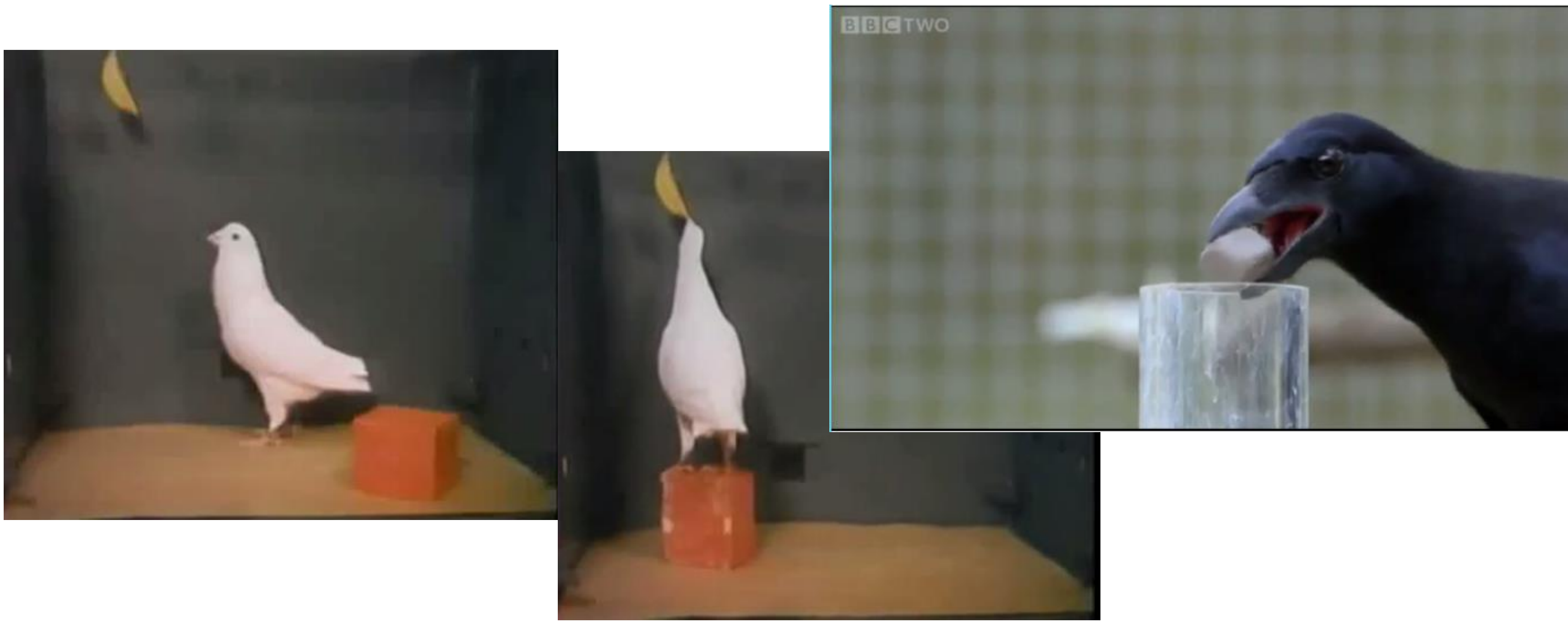
# AI as “symbol manipulation”

- The Logic Theorist, 1955
  - ➔ see lectures on logic
- The Dartmouth conference ("birth of AI”, 1956)
- List processing (Information Processing Language, IPL)
- Means-ends analysis ("reasoning as search")
  - ➔ see lectures on planning
- The General Problem Solver
- Heuristics to limit the search space
  - ➔ see lecture on informed search
- The physical symbol systems hypothesis
  - intelligent behavior can be reduced to/emulated by symbol manipulation
- The unified theory of cognition (1990, cognitive architectures: Soar, ACT-R)

- Newel&Simon: Computer science as empirical inquiry: symbols and search, 1975

# AI as "symbol manipulation"?

- The Box and Banana problem
  - Human, monkey, pigeon, crow..



# Problem-solving agent

- Four general steps in problem solving:
  - Goal formulation
    - What are the successful world states
  - Problem formulation
    - What actions and states to consider give the goal
  - Search
    - Determine the possible sequence of actions that lead to the states of known values and then choosing the best sequence.
  - Execute
    - Give the solution perform the actions.

# Problem types

- Deterministic, fully observable ⇒ *single state problem*
  - Agent knows exactly which state it will be in; solution is a sequence.
- Partial knowledge of states and actions:
- Non-observable ⇒ *sensorless or conformant problem*
  - Agent may have no idea where it is; solution (if any) is a sequence.
- Nondeterministic and/or partially observable ⇒ *contingency problem*
  - Percepts provide *new* information about current state; solution is a tree or policy; often interleave search and execution.
- Unknown state space ⇒ *exploration problem* ("online")
  - When states and actions of the environment are unknown.

# Problem formulation

- A problem is defined by:
  - An initial state, e.g. *Arad*
  - Successor function *S(X)*= set of action-state pairs
    - e.g. $S(Arad)=\{<Arad \rightarrow Zerind, Zerind>,\ldots\}$

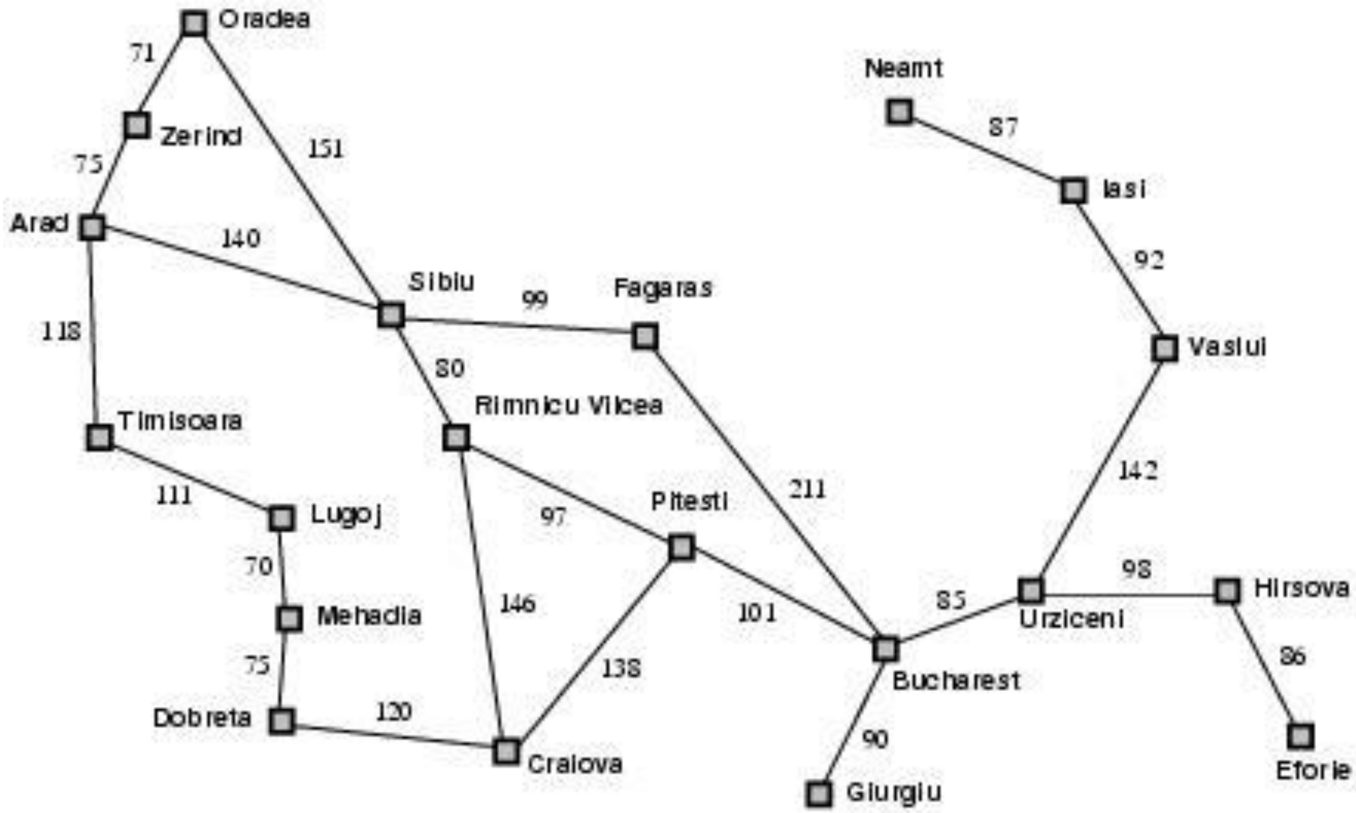
  intial state + successor function = state space
  - Goal test, can be
    - Explicit, e.g. *x='at bucharest'*
    - Implicit, e.g. *checkmate(x)*
  - Path cost (additive)
    - e.g. sum of distances, number of actions executed, …
    - *c(x,a,y)* is the step cost, assumed to be >= 0

A solution is a sequence of actions from initial to goal state.
Optimal solution has the lowest path cost.

# 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

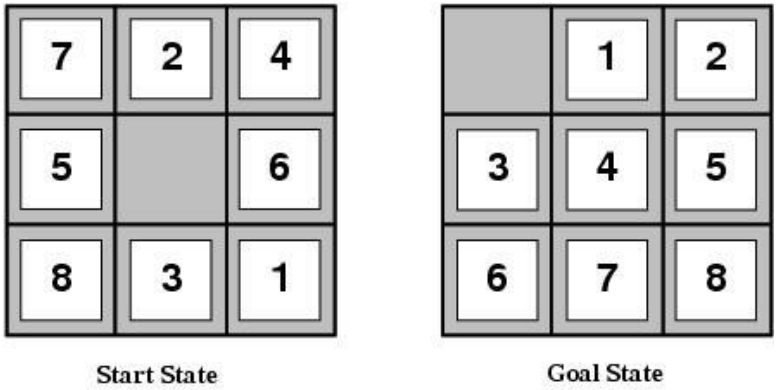


# Selecting a state space

- Real world is absurdly complex.

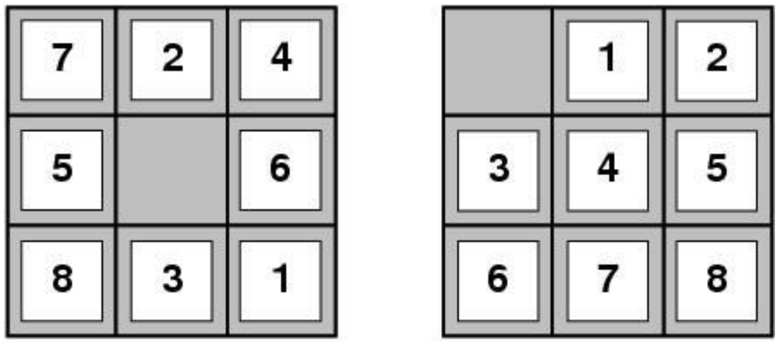
  State space must be *abstracted* for problem solving.
- (Abstract) state = set of real states.
- (Abstract) action = complex combination of real actions.
  - e.g. Arad $\rightarrow$Zerind represents a complex set of possible routes, detours, rest stops, etc.
  - The abstraction is valid if the path between two states is reflected in the real world.
- (Abstract) solution = set of real paths that are solutions in the real world.
- Each abstract action should be "easier" than the real problem.

# Example: 8-puzzle




- States??
- Initial state??
- Actions??
- Goal test??
- Path cost??

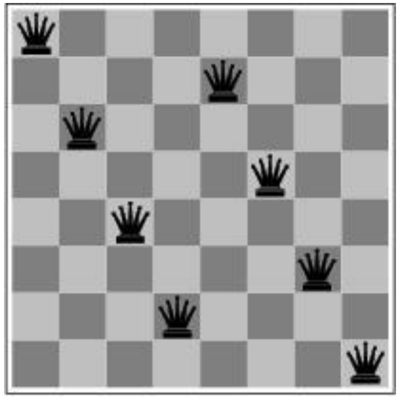
# Example: 8-puzzle




Start State    Goal State

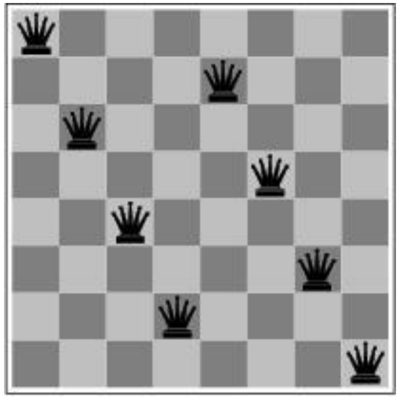
- States?? Integer location of each tile
- Initial state?? Any state can be initial
- Actions?? {*Left*, *Right*, *Up*, *Down*}
- Goal test?? Check whether goal configuration is reached
- Path cost?? Number of actions to reach goal

# Example: 8-queens problem



- States??
- Initial state??
- Actions??
- Goal test??
- Path cost??

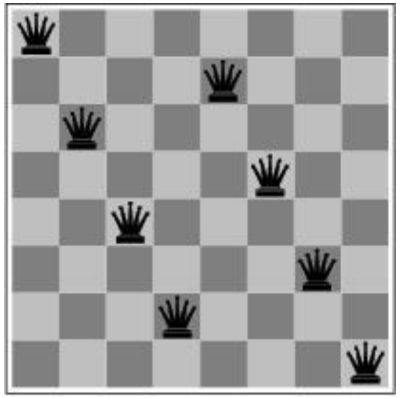
# Example: 8-queens problem



Incremental formulation vs. complete-state formulation

- States??
- Initial state??
- Actions??
- Goal test??
- Path cost??

# Example: 8-queens problem

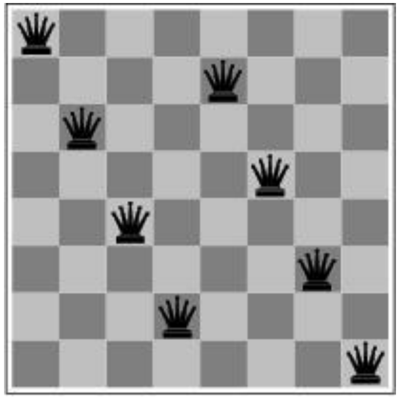


Incremental formulation

- States?? Any arrangement of 0 to 8 queens on the board
- Initial state?? No queens
- Actions?? Add queen in empty square
- Goal test?? 8 queens on board and none attacked
- Path cost?? None

  $3 \times 10^{14}$ possible sequences to investigate

# Example: 8-queens problem

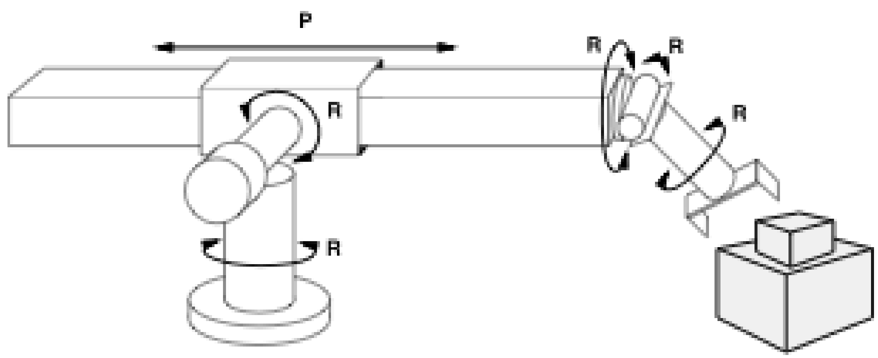


Incremental formulation (alternative)

- States?? $n$ ($0 \leq n \leq 8$) queens on the board, one per column in the $n$ leftmost columns with no queen attacking another.
- Actions?? Add queen in leftmost empty column such that is not attacking other queens

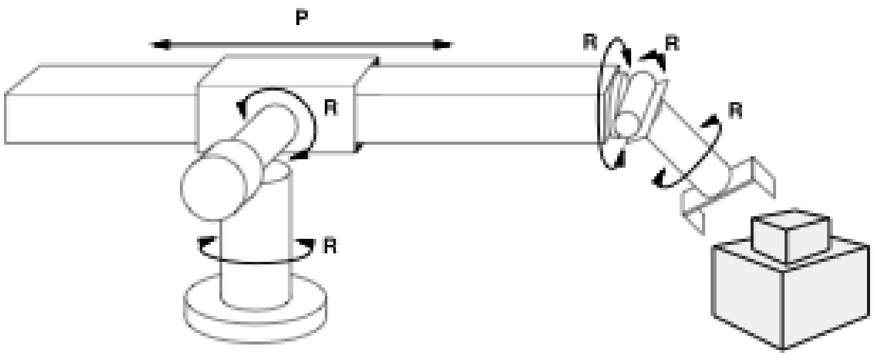
  2057 possible sequences to investigate; Yet makes no difference when $n$=100

# Example: robot assembly




- States??
- Initial state??
- Actions??
- Goal test??
- Path cost??

# Example: robot assembly

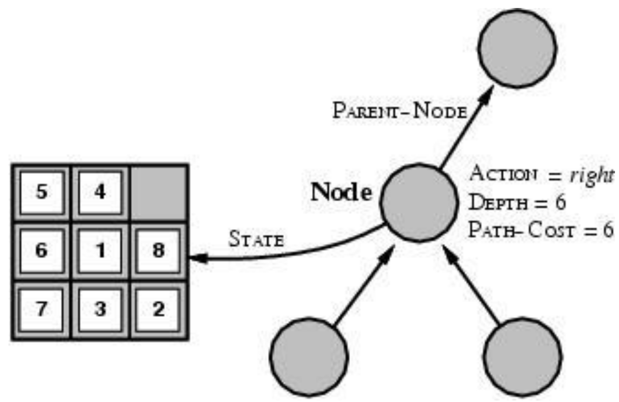



- States?? Real-valued coordinates of robot joint angles; parts of the object to be assembled.
- Initial state?? Any arm position and object configuration.
- Actions?? Continuous motion of robot joints
- Goal test?? Complete assembly (without robot)
- Path cost?? Time to execute

# Basic search algorithms

- How do we find the solutions of previous problems?
  - Search the state space (remember complexity of space depends on state representation)
  - Here: search through *explicit tree generation*
    - ROOT= initial state.
    - Nodes and leafs generated through successor function.
  - In general search generates a graph (same state through multiple paths)

# State space vs. search tree




- A *state* is a (representation of) a physical configuration
- A *node* is a data structure belong to a search tree
  - A node has a parent, children, … and ncludes path cost, depth, …
  - Here *node= <state, parent-node, action, path-cost, depth>*
  - *FRINGE*= contains generated nodes which are not yet expanded.
    - White nodes with black outline

# Tree search algorithm

```
function TREE-SEARCH(problem,fringe) return a solution or failure
   fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
   loop do
         if EMPTY?(fringe) then return failure
         node ← REMOVE-FIRST(fringe)
         if GOAL-TEST[problem] applied to STATE[node] succeeds
                  then return SOLUTION(node)
         fringe ← INSERT-ALL(EXPAND(node, problem), fringe)
```

# Tree search algorithm (2)

**function** EXPAND(*node,problem*) **return** a set of nodes

*successors* $\leftarrow$ the empty set

**for each** <*action*, *result*> **in** SUCCESSOR-FN[*problem*](STATE[*node*]) **do**

*s* $\leftarrow$ a new NODE

STATE[*s*] $\leftarrow$ *result*

PARENT-NODE[*s*] $\leftarrow$ *node*

ACTION[*s*] $\leftarrow$ *action*

PATH-COST[*s*] $\leftarrow$ PATH-COST[*node*] + STEP-COST(*node, action,s*)

DEPTH[*s*] $\leftarrow$ DEPTH[*node*]+1

add *s* to *successors*

**return** *successors*

# Search strategies

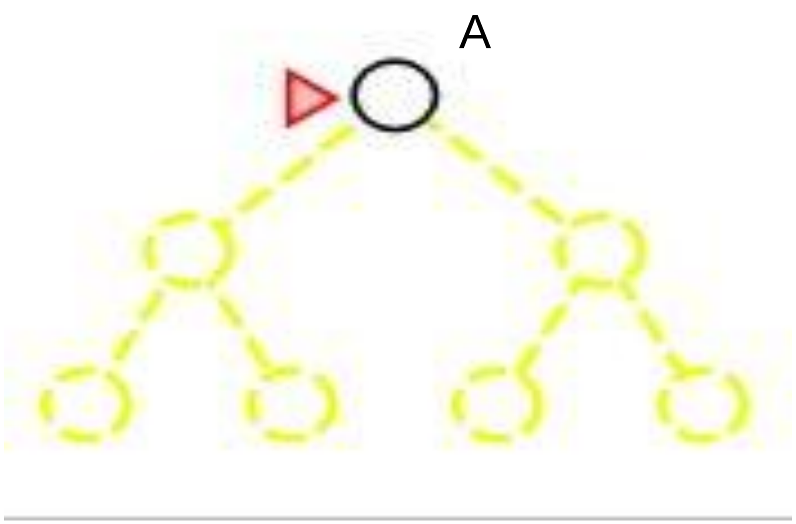
- A strategy is defined by picking the order of node expansion.
- Problem-solving performance is measured in four ways:
  - Completeness; *Does it always find a solution if one exists?*
  - Optimality; *Does it always find the least-cost solution?*
  - Space Complexity; *Number of nodes stored in memory during search?*
  - Time Complexity; *Number of nodes generated/expanded?*
- Time and space complexity are measured in terms of problem difficulty defined by:
  - *b - maximum branching factor of the search tree*
  - *d - depth of the least-cost solution*
  - *m - maximum depth of the state space (may be ∞)*

# Uninformed search strategies

- (a.k.a. blind search) = use only information available in problem definition.
  - When strategies can determine whether one non-goal state is better than another $\rightarrow$ informed search.
- Categories defined by expansion algorithm:
  - Breadth-first search
  - Uniform-cost search
  - Depth-first search
  - Depth-limited search
  - Iterative deepening search.
  - Bidirectional search

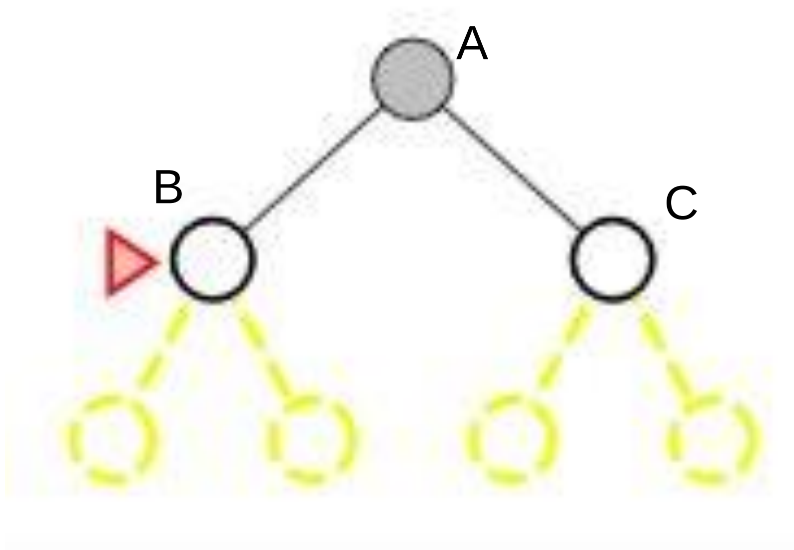
# BF-search, an example

- Expand *shallowest* unexpanded node
- Implementation: *fringe* is a FIFO queue



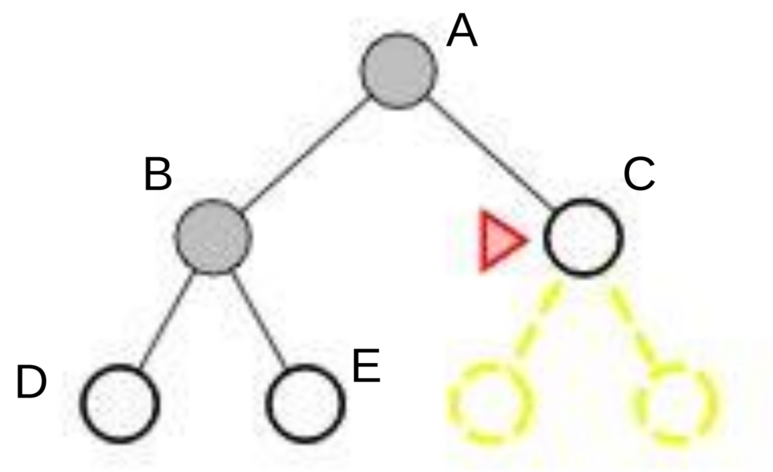
# BF-search, an example

- Expand *shallowest* unexpanded node
- Implementation: *fringe* is a FIFO queue



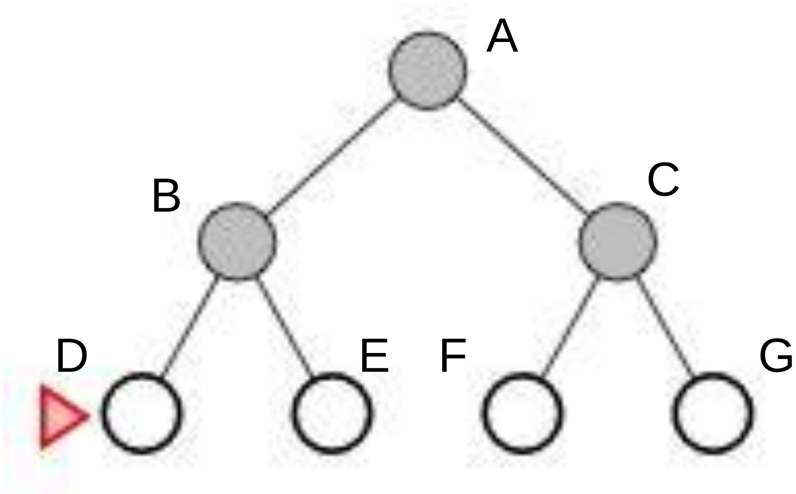
# BF-search, an example

- Expand *shallowest* unexpanded node
- Implementation: *fringe* is a FIFO queue



# BF-search, an example

- Expand *shallowest* unexpanded node
- Implementation: *fringe* is a FIFO queue



# BF-search; evaluation

- Completeness:
  - *Does it always find a solution if one exists?*
  - YES
    - If shallowest goal node is at some finite depth d
    - Condition: If *b* is finite
      - (maximum num. Of succ. nodes is finite)

# BF-search; evaluation

- Completeness:
  - YES (if $b$ is finite)
- Time complexity:
  - Assume a state space where every state has $b$ successors.
    - root has $b$ successors, each node at the next level has again b successors (total $b^2$), …
    - Assume solution is at depth $d$
    - Worst case; expand all but the last node at depth $d$
    - Total numb. of nodes generated:

$$b + b^2 + b^3 + \ldots + b^d + (b^{d+1} - b) = O(b^{d+1})$$

# BF-search; evaluation

- Completeness:
  - YES (if *b* is finite)
- Time complexity:
  - Total numb. of nodes generated:

- Space complexity:
  - Idem if each node is retained in memory

$$b + b^2 + b^3 + \ldots + b^d + (b^{d+1} - b) = O(b^{d+1})$$

# BF-search; evaluation

- Completeness:
  - YES (if *b* is finite)
- Time complexity:
  - Total numb. of nodes generated:
- Space complexity:
  - Idem if each node is retained in memory

$$b + b^2 + b^3 + \ldots + b^d + (b^{d+1} - b) = O(b^{d+1})$$

- Optimality:
  - *Does it always find the least-cost solution?*
  - In general YES
    - unless actions have different cost.

# BF-search; evaluation

- Two lessons:
  - Memory requirements are a bigger problem than its execution time.
  - Exponential complexity search problems cannot be solved by uninformed search methods for any but the smallest instances.

| DEPTH2 | NODES | TIME | MEMORY |
|---|---|---|---|
| 2 | 1100 | 0.11 seconds | 1 megabyte |
| 4 | 111100 | 11 seconds | 106 megabytes |
| 6 | $10^7$ | 19 minutes | 10 gigabytes |
| 8 | $10^9$ | 31 hours | 1 terabyte |
| 10 | $10^{11}$ | 129 days | 101 terabytes |
| 12 | $10^{13}$ | 35 years | 10 petabytes |
| 14 | $10^{15}$ | 3523 years | 1 exabyte |

# Uniform-cost search

- Extension of BF-search:
  - Expand node with *lowest path cost*
- Implementation: *fringe* = queue ordered by path cost.

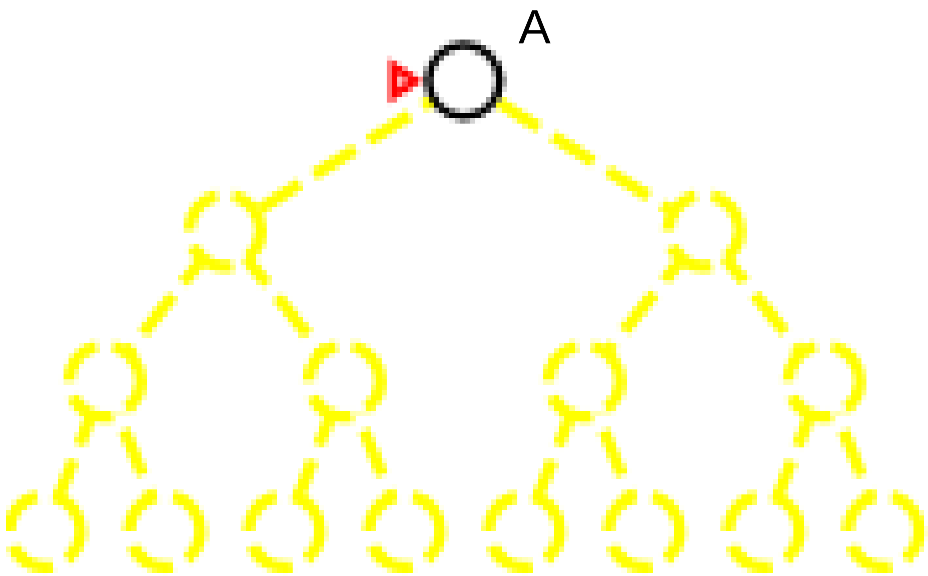
- UC-search is the same as BF-search when all step-costs are equal.

# Uniform-cost search

- Completeness:
  - YES, if step-cost > $\varepsilon$ (smal positive constant)
- Time complexity:
  - Assume C* the cost of the optimal solution.
  - Assume that every action costs at least $\varepsilon$
  - Worst-case:
- Space complexity:
  - Idem to time complexity $O(b^{C^*/\varepsilon})$
- Optimality:
  - nodes expanded in order of increasing path cost.
  - YES, if complete.

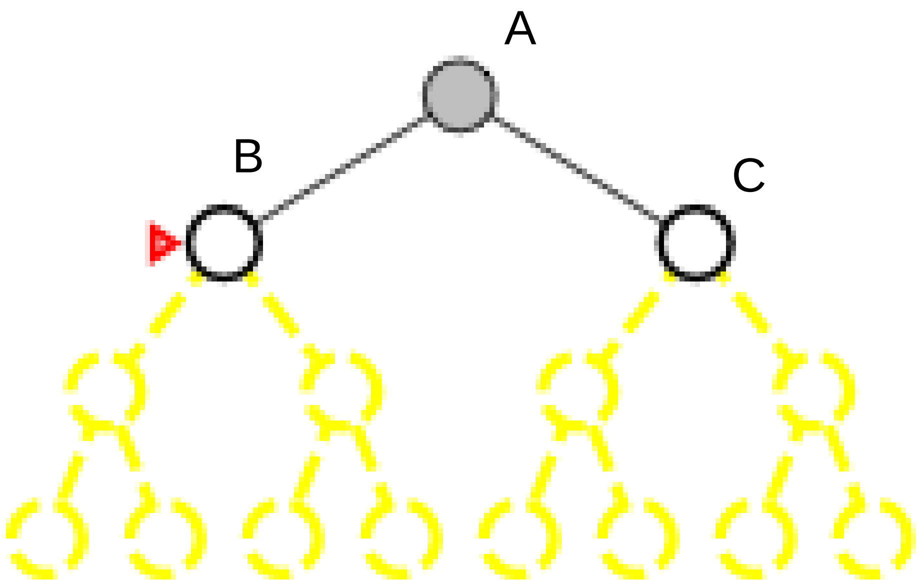
# DF-search, an example

- Expand *deepest* unexpanded node
- Implementation: *fringe* is a LIFO queue (=stack)



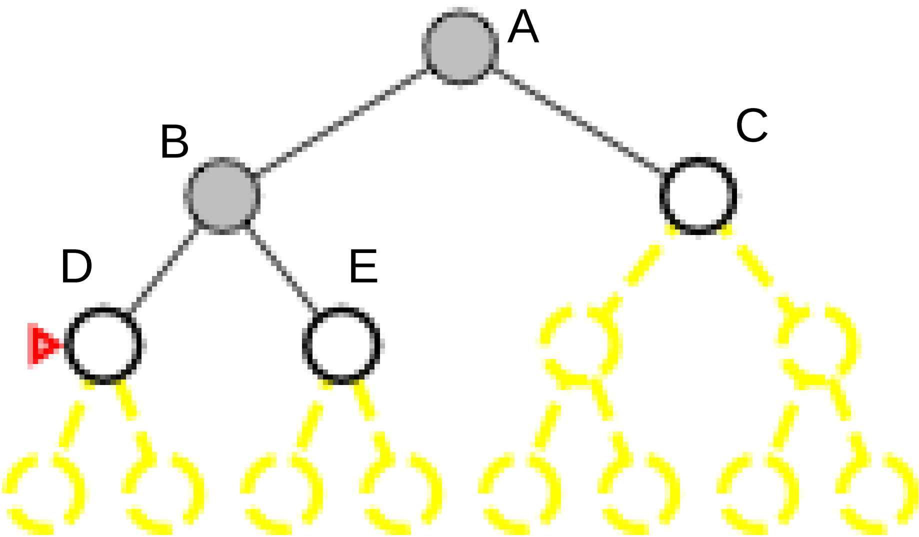
# DF-search, an example

- Expand *deepest* unexpanded node
- Implementation: *fringe* is a LIFO queue (=stack)



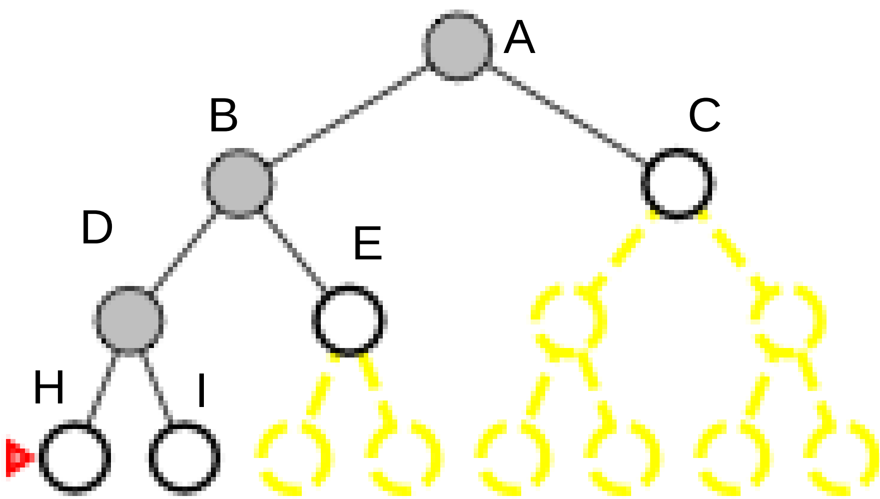
# DF-search, an example

- Expand *deepest* unexpanded node
- Implementation: *fringe* is a LIFO queue (=stack)



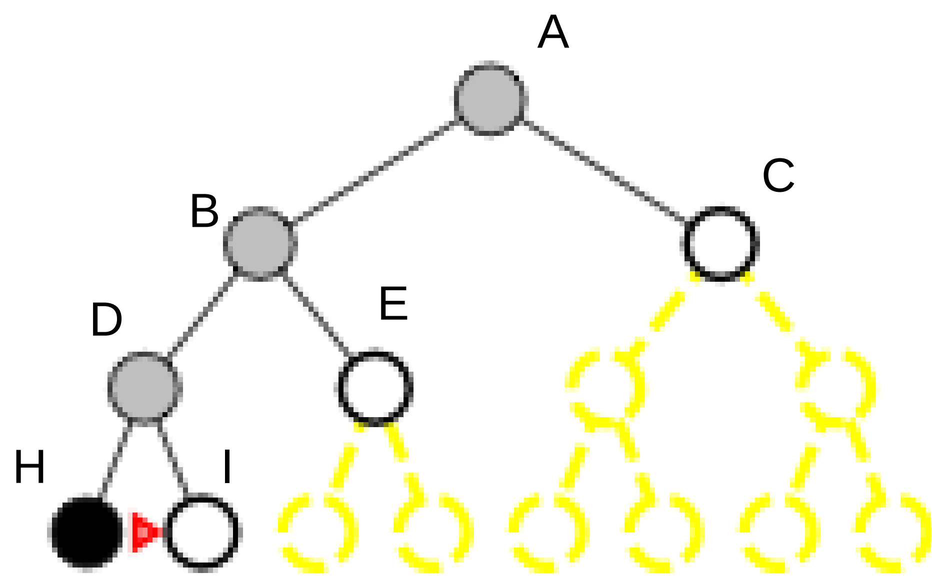
# DF-search, an example

- Expand *deepest* unexpanded node
- Implementation: *fringe* is a LIFO queue (=stack)



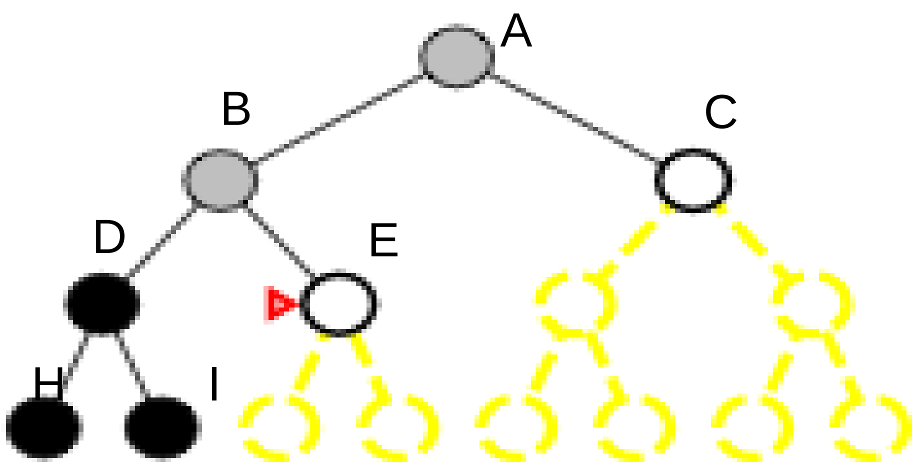
# DF-search, an example

- Expand *deepest* unexpanded node
- Implementation: *fringe* is a LIFO queue (=stack)



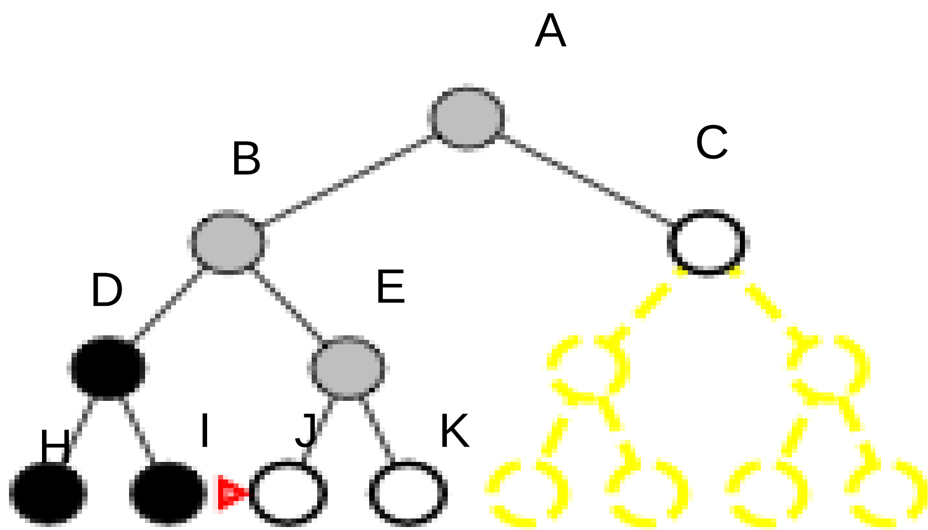
# DF-search, an example

- Expand *deepest* unexpanded node
- Implementation: *fringe* is a LIFO queue (=stack)



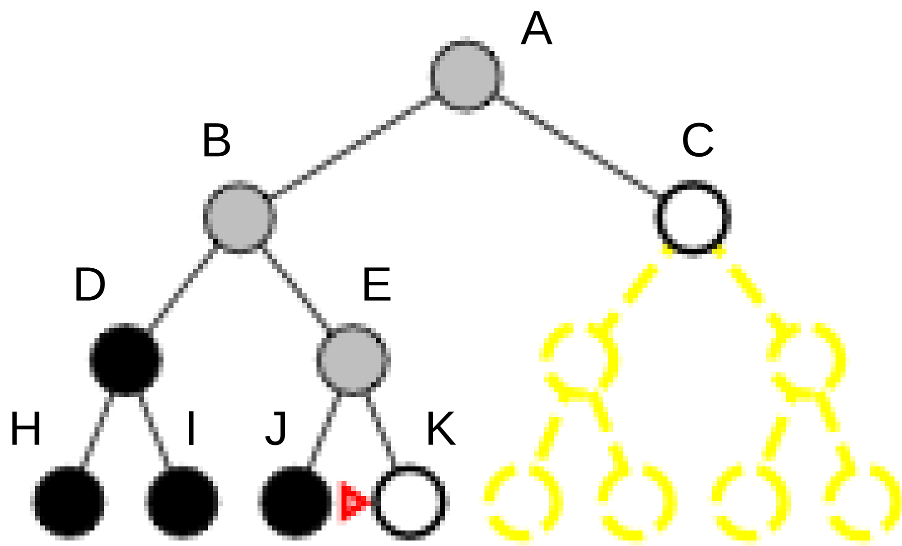
# DF-search, an example

- Expand *deepest* unexpanded node
- Implementation: *fringe* is a LIFO queue (=stack)



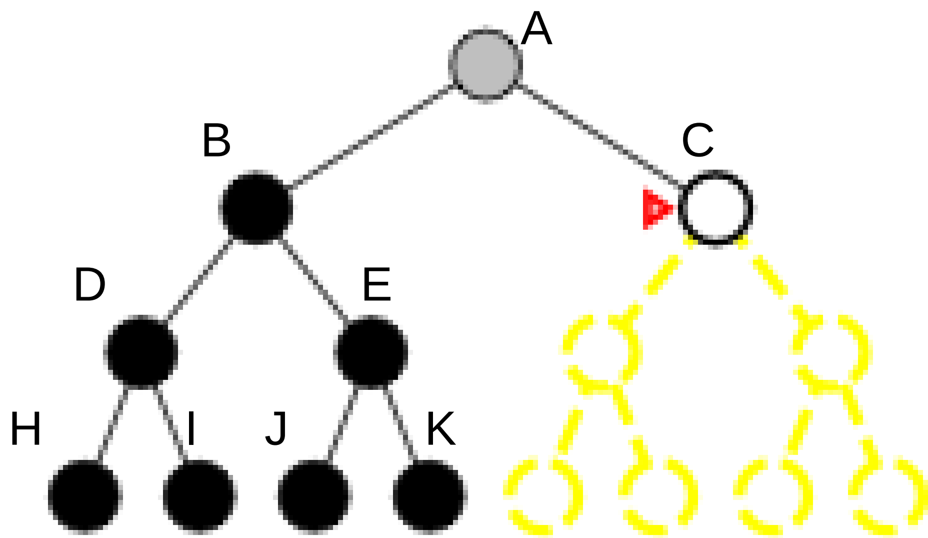
# DF-search, an example

- Expand *deepest* unexpanded node
- Implementation: *fringe* is a LIFO queue (=stack)



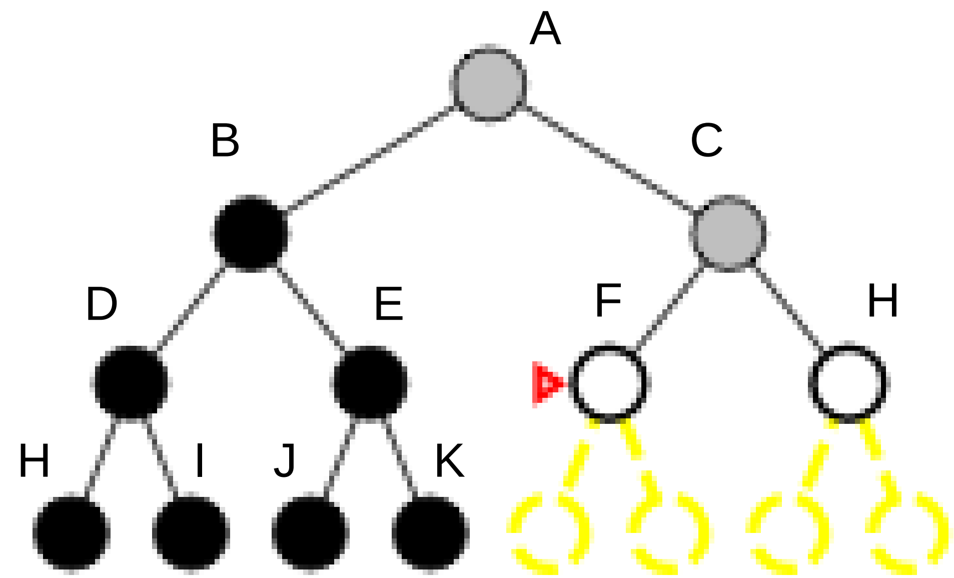
# DF-search, an example

- Expand *deepest* unexpanded node
- Implementation: *fringe* is a LIFO queue (=stack)



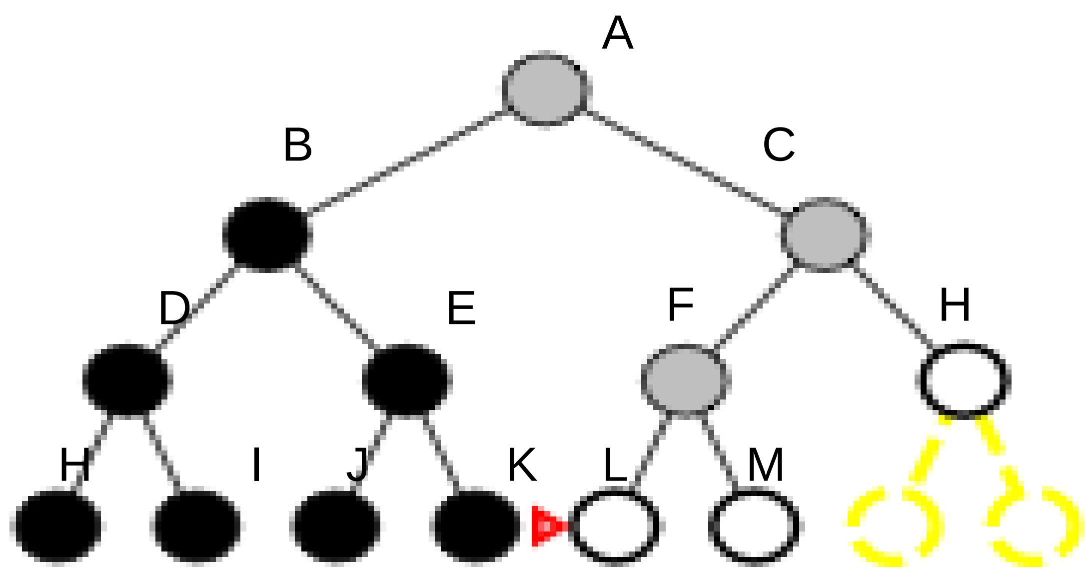
# DF-search, an example

- Expand *deepest* unexpanded node
- Implementation: *fringe* is a LIFO queue (=stack)



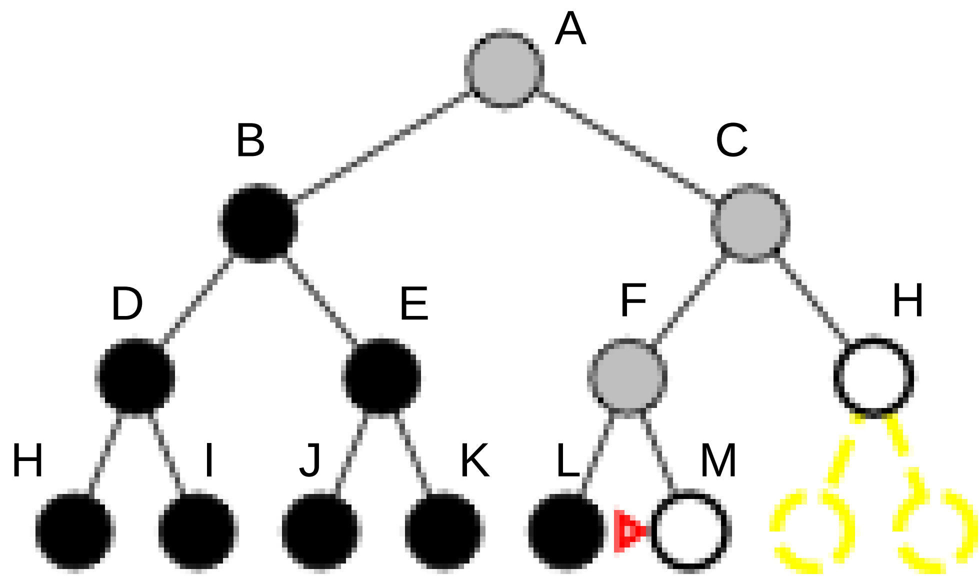
# DF-search, an example

- Expand *deepest* unexpanded node
- Implementation: *fringe* is a LIFO queue (=stack)



# DF-search, an example

- Expand *deepest* unexpanded node
- Implementation: *fringe* is a LIFO queue (=stack)



# DF-search; evaluation

- Completeness;
  - *Does it always find a solution if one exists?*
  - NO
    - *unless* search space is finite and no loops are possible.

# DF-search; evaluation

- Completeness;
  - NO unless search space is finite.
- Time complexity;
  - Terrible if *m* is much larger than *d* (depth of optimal solution) $O(b^m)$
  - But if many solutions, then faster than BF-search

# DF-search; evaluation

- Completeness;
  - NO unless search space is finite.
- Time complexity; $O(b^m)$
- Space complexity; $O(bm+1)$
  - Backtracking search uses even less memory
    - One successor instead of all *b.*

# DF-search; evaluation

- Completeness;
  - NO unless search space is finite.
- Time complexity; $O(b^m)$
- Space complexity; $O(bm+1)$
- Optimallity; No
  - Same issues as completeness
  - Assume node J and C contain goal states

# Depth-limited search

- Is DF-search with depth limit $l$.
  - i.e. nodes at depth $l$ have no successors.
  - Problem knowledge can be used
- Solves the infinite-path problem.
- If $l < d$ then incompleteness results.
- If $l > d$ then not optimal.
- Time complexity: $O(b^l)$
- Space complexity: $O(bl)$

# Depth-limited algorithm

**function** DEPTH-LIMITED-SEARCH(*problem,limit*) **return** a solution or failure/cutoff

**return** RECURSIVE-DLS(MAKE-NODE(INITIAL-STATE[*problem*]),*problem*,*limit*)

**function** RECURSIVE-DLS(*node*, *problem, limit*) **return** a solution or failure/cutoff

*cutoff_occurred?* ← false

**if** GOAL-TEST[*problem*](STATE[*node*]) **then return** SOLUTION(*node*)

**else if** DEPTH[*node*] == *limit* then return *cutoff*

**else for each** *successor* **in** EXPAND(*node*, *problem*) **do**

*result* ← RECURSIVE-DLS(*successor*, *problem*, *limit*)

**if** *result == cutoff* **then** *cutoff_occurred?* ← true

**else if** *result ≠ failure* **then return** *result*

**if** *cutoff_occurred?* **then return** *cutoff* **else return** failure

# Iterative deepening search

- What?
  - A general strategy to find best depth limit *l*.
    - Goals is found at depth *d*, the depth of the shallowest goal-node.
  - Often used in combination with DF-search
- Combines benefits of DF- en BF-search

# Iterative deepening search

**function** ITERATIVE_DEEPENING_SEARCH(*problem*) **return** a solution or failure

**inputs:** *problem*

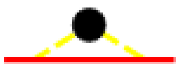
**for** *depth* ← 0 to ∞ **do**

*result* ← DEPTH-LIMITED_SEARCH(*problem*, *depth*)

**if** *result* ≠ *cuttoff* **then return** *result*

# ID-search, example

- Limit=0

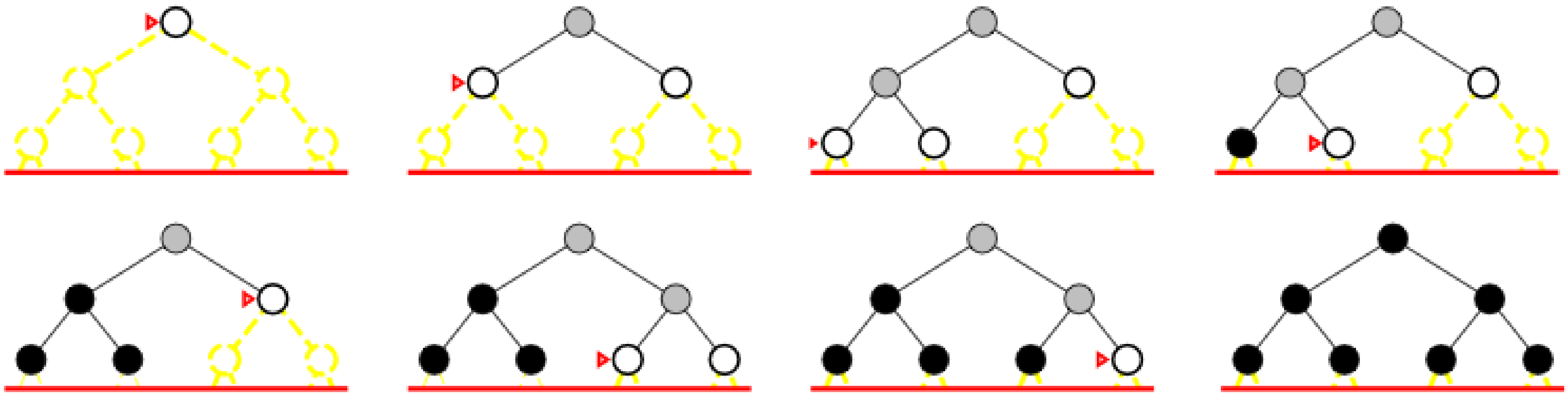


# ID-search, example

- Limit=1

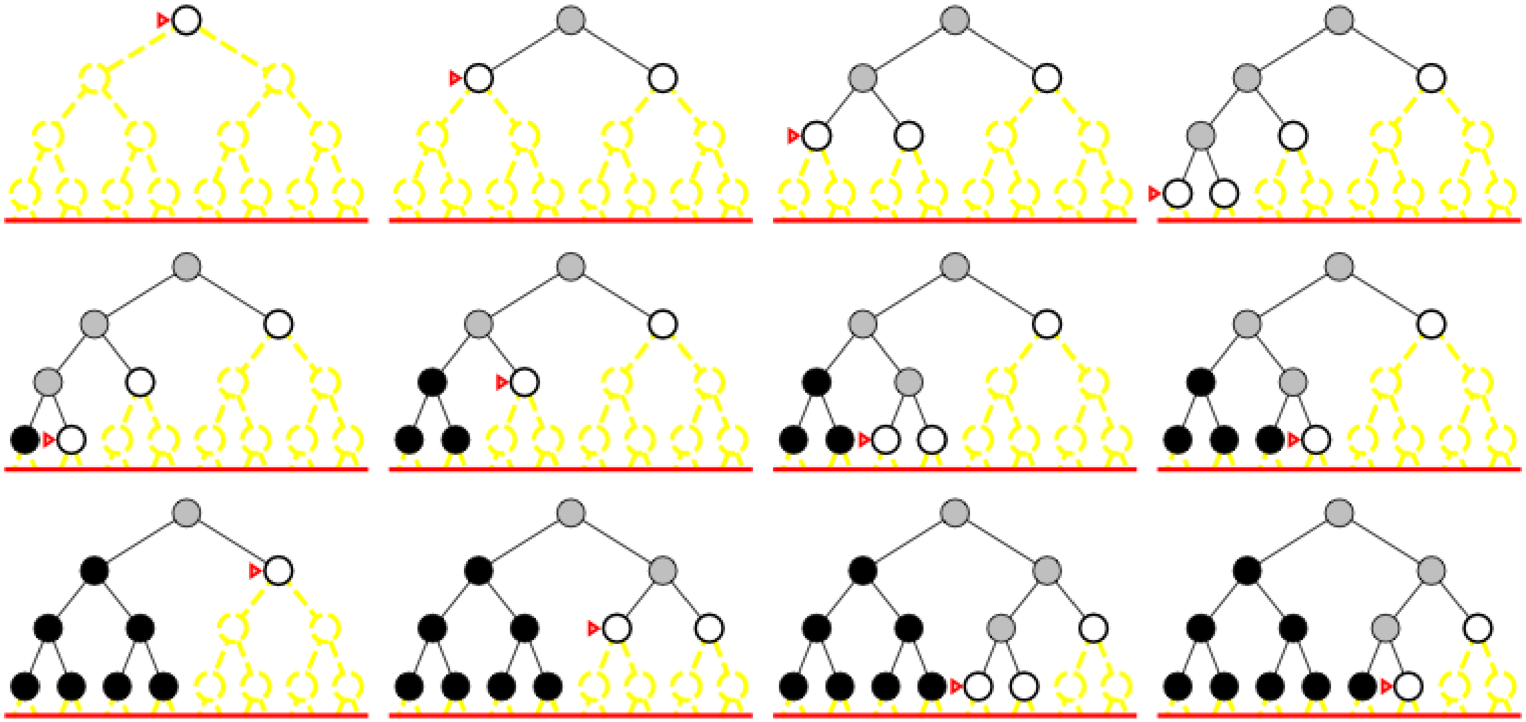
# ID-search, example

- Limit=2



# ID-search, example

- Limit=3



# ID search, evaluation

- Completeness:
  - YES (no infinite paths)

# ID search, evaluation

- Completeness:
  - YES (no infinite paths)
- Time complexity:
  - Algorithm seems costly due to repeated generation of certain states.
  - Node generation:
    - level d: once
    - level d-1: 2
    - level d-2: 3
    - …
    - level 2: d-1
    - level 1: d

$$O(b^d)$$

$$N(IDS) = (d)b + (d-1)b^2 + \ldots + (1)b^d$$

$$N(BFS) = b + b^2 + \ldots + b^d + (b^{d+1} - b)$$

Num. Comparison for b=10 and d=5 solution at far right

$$N(IDS) = 50 + 400 + 3000 + 20000 + 100000 = 123450$$

$$N(BFS) = 10 + 100 + 1000 + 10000 + 100000 + 999990 = 1111100$$

# ID search, evaluation

- Completeness:
  - YES (no infinite paths)
- Time complexity: $O(b^d)$
- Space complexity: $O(bd)$
  - Cfr. depth-first search

# ID search, evaluation

- Completeness:
  - YES (no infinite paths)
- Time complexity: $O(b^d)$
- Space complexity: $O(bd)$
- Optimality:
  - YES if step cost is 1.
  - Can be extended to iterative lengthening search
    - Same idea as uniform-cost search
    - Increases overhead.

# Summary of algorithms

| Criterion | Breadth-First | Uniform-cost | Depth-First | Depth-limited | Iterative deepening | Bidirectional search |
|---|---|---|---|---|---|---|
| Complete? | YES* | YES* | NO | YES, if $l \geq d$ | YES | YES* |
| Time | $b^{d+1}$ | $b^{C^*/e}$ | $b^m$ | $b^l$ | $b^d$ | $b^{d/2}$ |
| Space | $b^{d+1}$ | $b^{C^*/e}$ | $bm$ | $bl$ | $bd$ | $b^{d/2}$ |
| Optimal? | YES* | YES* | NO | NO | YES | YES |

# Summary

- The symbols&search paradigm in AI
- Uninformed search
  - Space complexity: OK!
  - Time complexity: exp. ➔ the knowledge paradigm in AI
- Suggested reading
  - Newel&Simon: Computer science as empirical inquiry: symbols and search, 1975
  - Cognitive architectures: ACT-R
    - http://act-r.psy.cmu.edu/
    - http://act-r.psy.cmu.edu/about/
    - Allen Newell describes cognitive architectures as the way to answer one of the ultimate scientific questions: "How can the human mind occur in the physical universe?
      - http://act-r.psy.cmu.edu/misc/newellclip.mpg