Introduction to Artificial Intelligence

Lecture: Uninformed Search

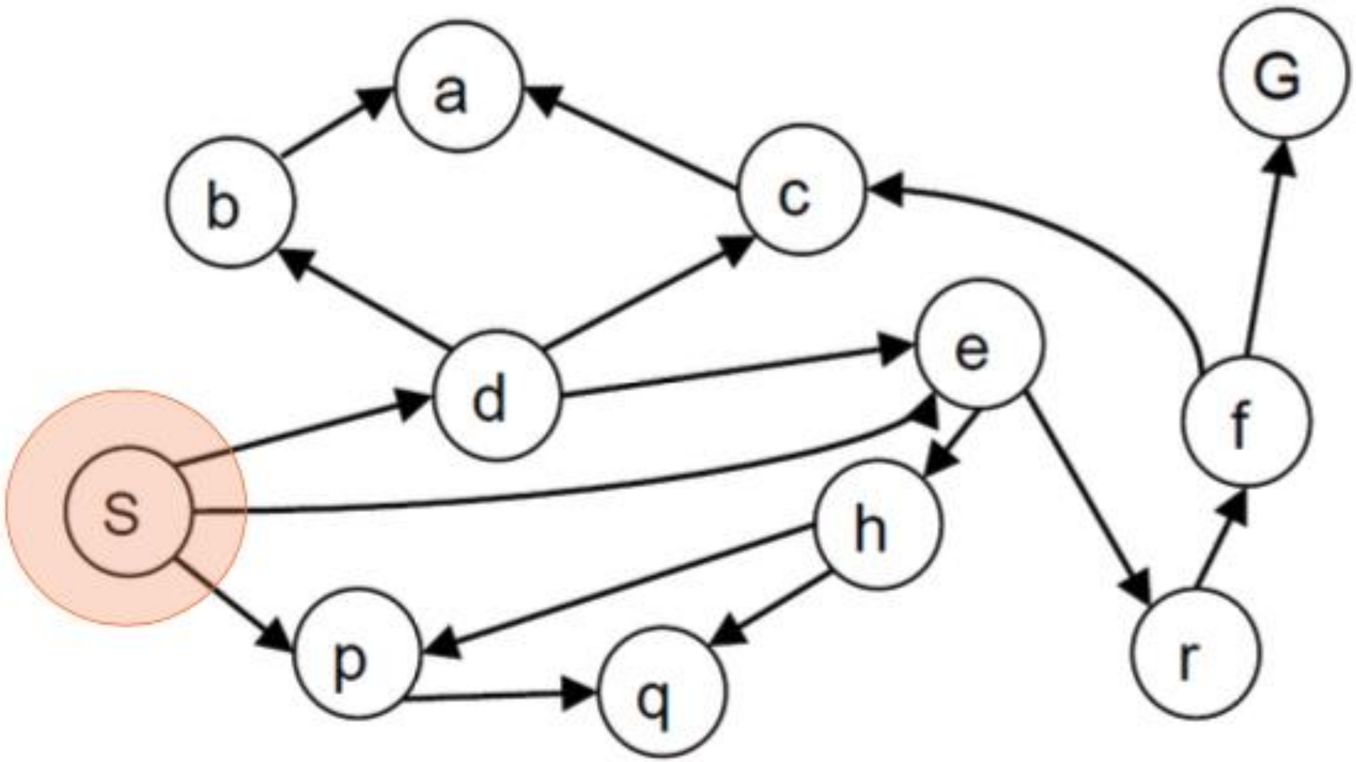
Outline

- Uninformed Search Strategies
- Breadth-first Search
- Uniform-cost Search
- Depth-first Search
- Depth-limited Search
- Iterative Deepening Search
- Bidirectional Search

Uninformed Search Strategies

- No additional information about states beyond that provided in the problem definition → Blind Search
  - Breadth-first Search
  - Uniform-cost Search
  - Depth-first Search
  - Depth-limited Search
  - Iterative Deepening Search
  - Bidirectional Search

Breadth-first Search



Breadth-first Search

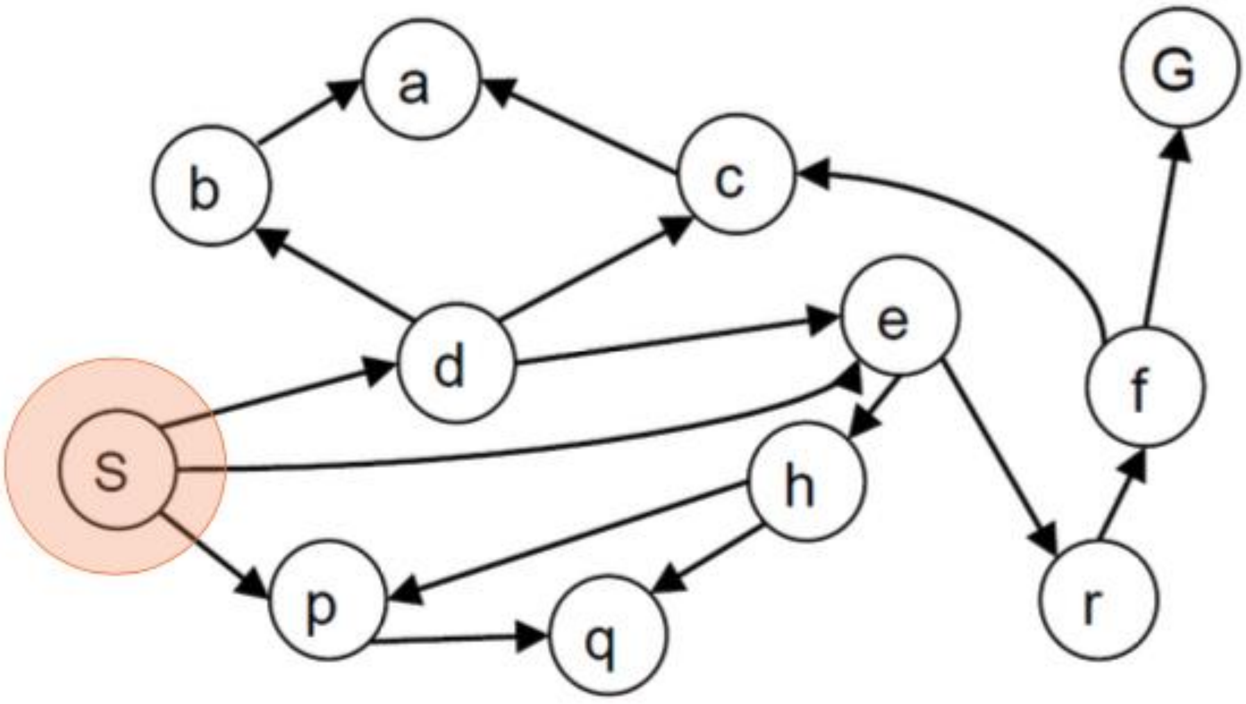
- Implementation: frontier is a FIFO queue
- The goal test is applied to each node when it is generated rather than when it is selected for expansion.
- Discard any new path to a state already in the frontier or in the explored set.

Breadth-first Search

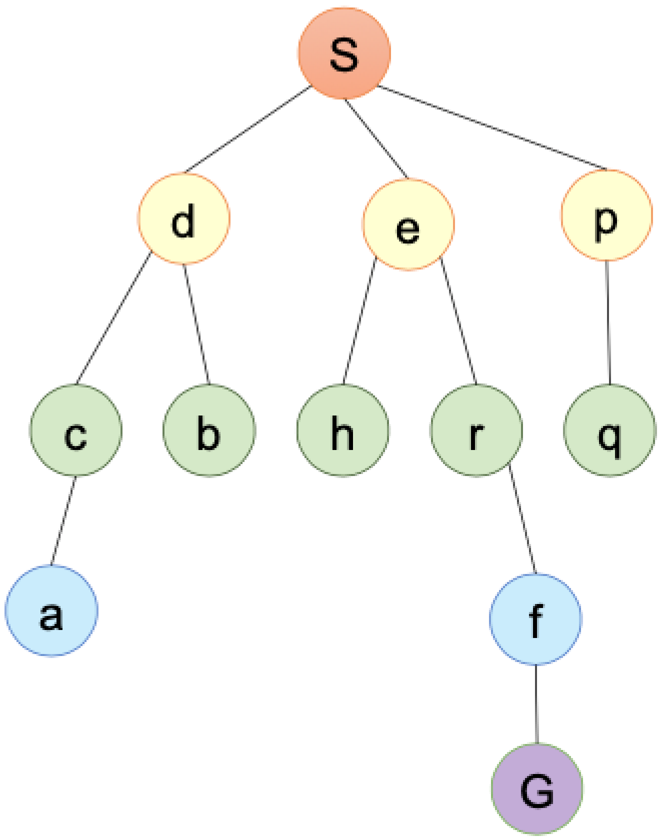
```
function BREADTH-FIRST-SEARCH(problem) returns a solution, or failure
    node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
    if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
    frontier ← a FIFO queue with node as the only element
    explored ← an empty set
    loop do
        if EMPTY?( frontier) then return failure
        node ← POP(frontier)
        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)
```

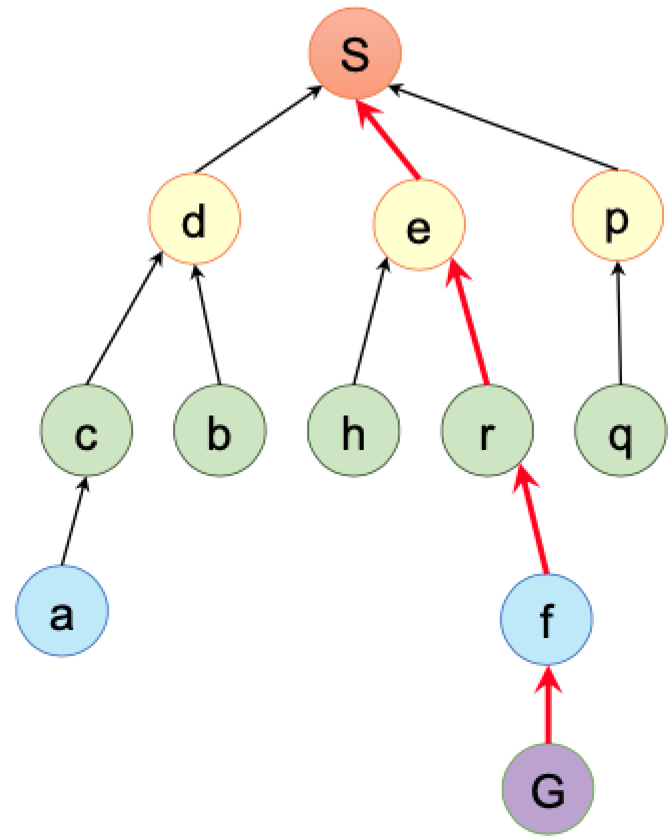
Breadth-first Search

- Draw the search tree S → G
- Write down the path
- Nodes at the same level are handled in the alphabetic order



Breadth-first Search





Breadth-first Search

- Evaluation
  - Completeness: YES
  - Time complexity: $O(b^d)$
  - Space complexity: $O(b^d)$
  - Optimality: YES if costs are uniform
- Terms:
  - b: maximal branching factor
  - d: level/depth of the solution
  - m: height of the search tree

Uniform-cost Search

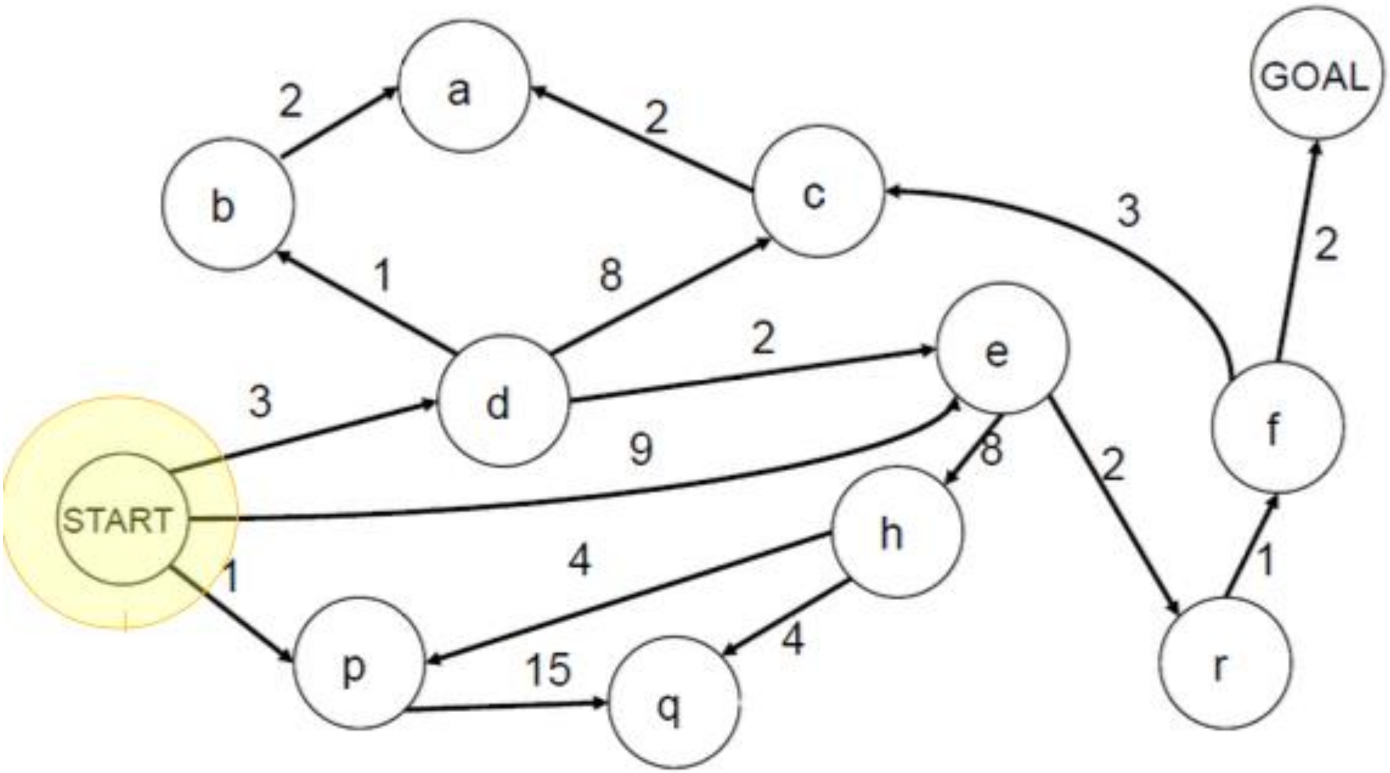
- UCS expands the node n with the lowest path cost $g(n)$
- Implementation: frontier is a priority queue ordered by g
  - Equivalent to breadth-first search if step costs all equal
  - Equivalent to Dijkstra's algorithm in general
- The goal test is applied to a node when it is selected for expansion.

Uniform-cost Search

```
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 element
    explored ← an empty set
    loop do
        if EMPTY?( frontier) then return failure
        node ← POP(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
```

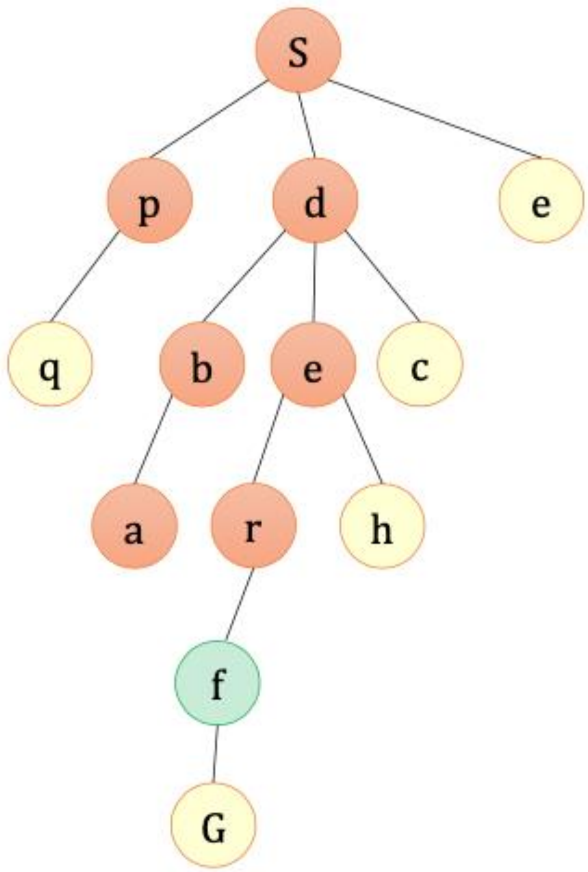
Uniform-cost Search

- Draw the search tree S → G
- Write down the path



Uniform-cost Search

- Search path: S → d → e → r → f → G, cost = 10



Uniform-cost Search

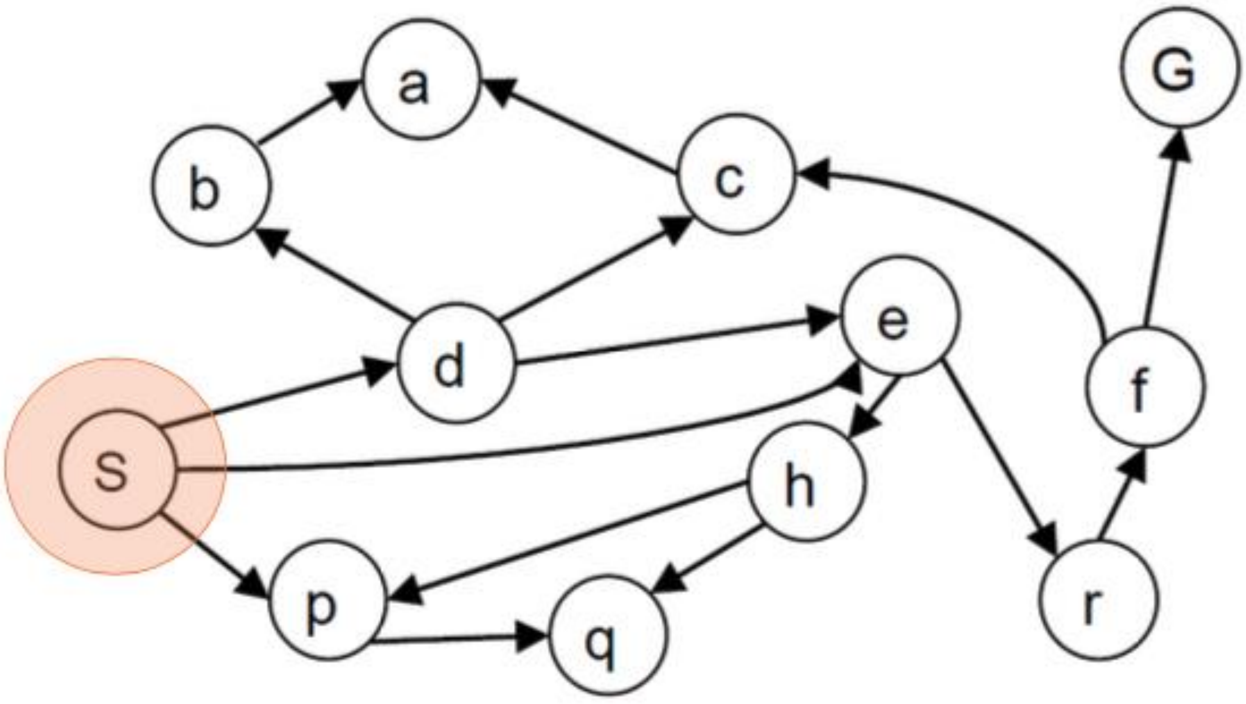
- Evaluation:
  - Completeness: YES if the best solution has a finite cost and minimum arc cost is positive.
  - Time complexity: $O(b^{1+ C*/\epsilon})$
    - $C*$: the cost of the best solution
    - ϵ : minimal action cost
  - Space complexity: $O(b^{1+ C*/\epsilon})$
  - Optimality: YES

Depth-first Search

- Implementation: frontier is a LIFO Stack
- Evaluation:
  - Completeness: YES if loops prevented
  - Time complexity: $O(b^m)$
  - Space complexity: $O(bm)$
  - Optimality: NO

Depth-first Search

- Draw the search tree S → G
- Write down the path
- Nodes at the same level are handled in the alphabetic order



Depth-limited Search

- Standard DFS with a predetermined depth limit l, i.e., nodes at depth l are treated as if they have no successors.

  $\rightarrow$ infinite problems solved

- Depth limits can be based on knowledge of the problem.

Depth-limited Search

```
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
```

Depth-limited Search

- Evaluation:
  - Completeness: maybe NO if $l < d$
  - Time complexity: $O(b^l)$
  - Space complexity: $O(bl)$
  - Optimality: NO if $l > d$

Iterative Deepening Search

- General strategy, often used in combination with depth-first tree search to find the best depth limit.
- Gradually increase the limit until a goal is found.

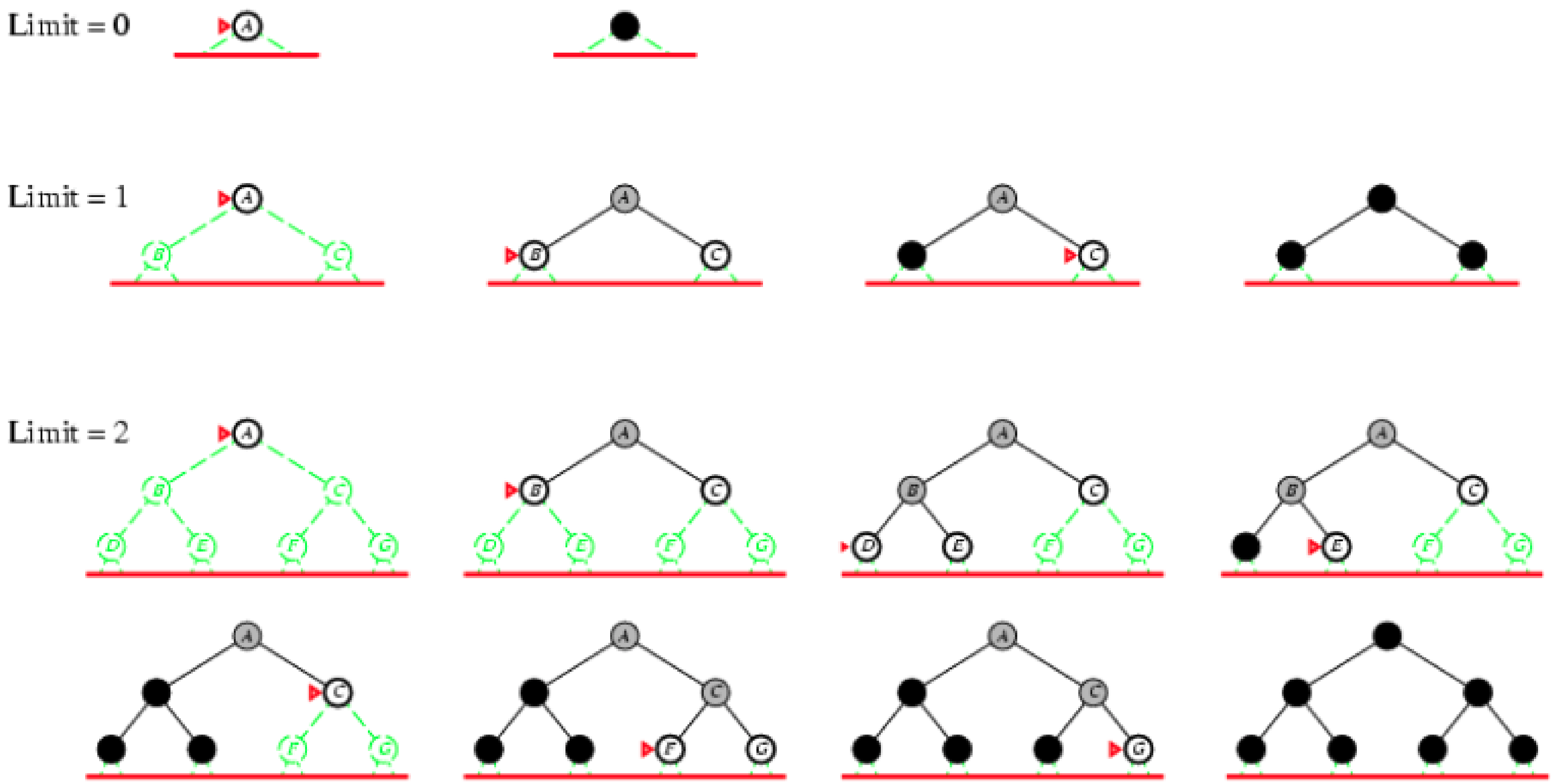
function ITERATIVE-DEEPENING-SEARCH(problem) **returns** a solution, or failure

    for depth = 0 **to** ∞ **do**

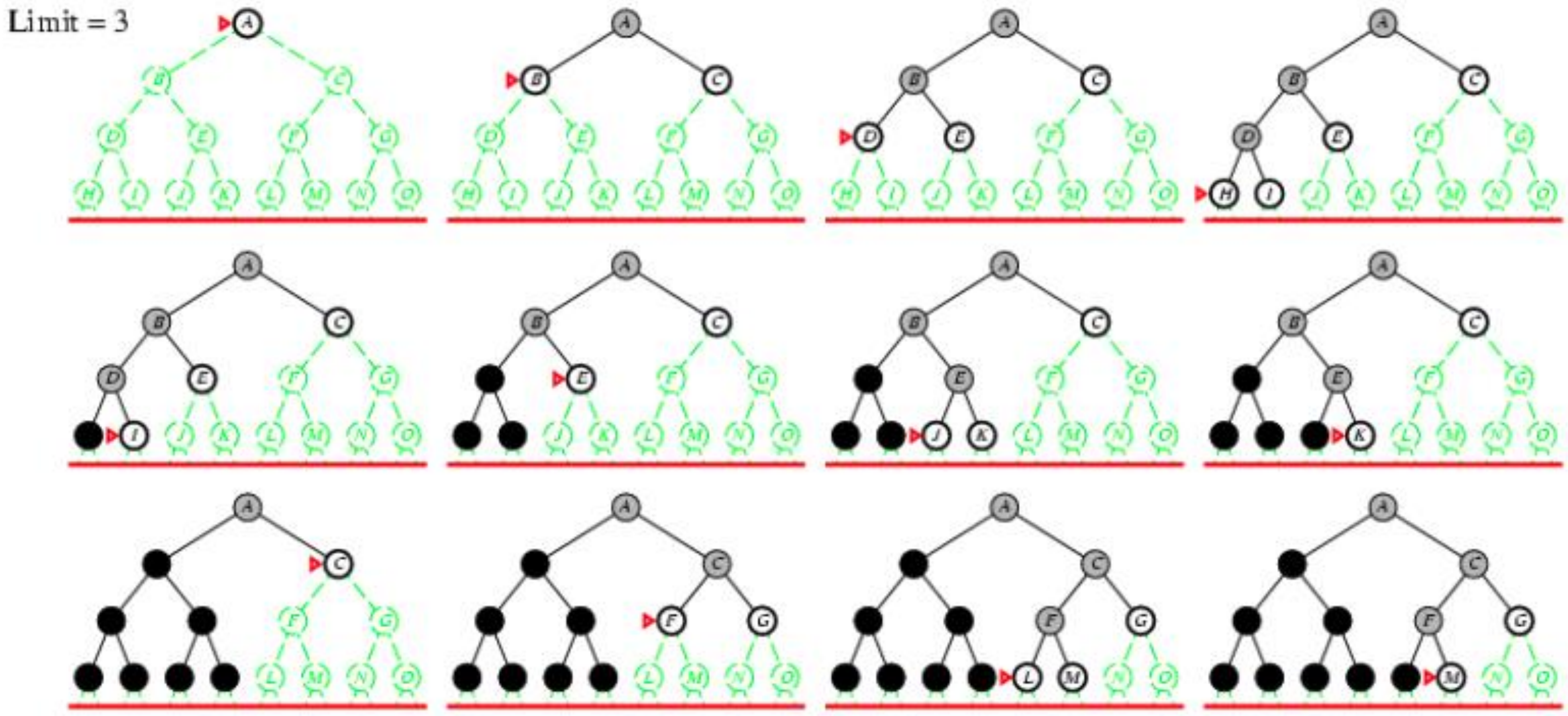
        result ← DEPTH-LIMITED-SEARCH(problem, depth)

        if result != cutoff **then return** result

Iterative Deepening Search



Iterative Deepening Search

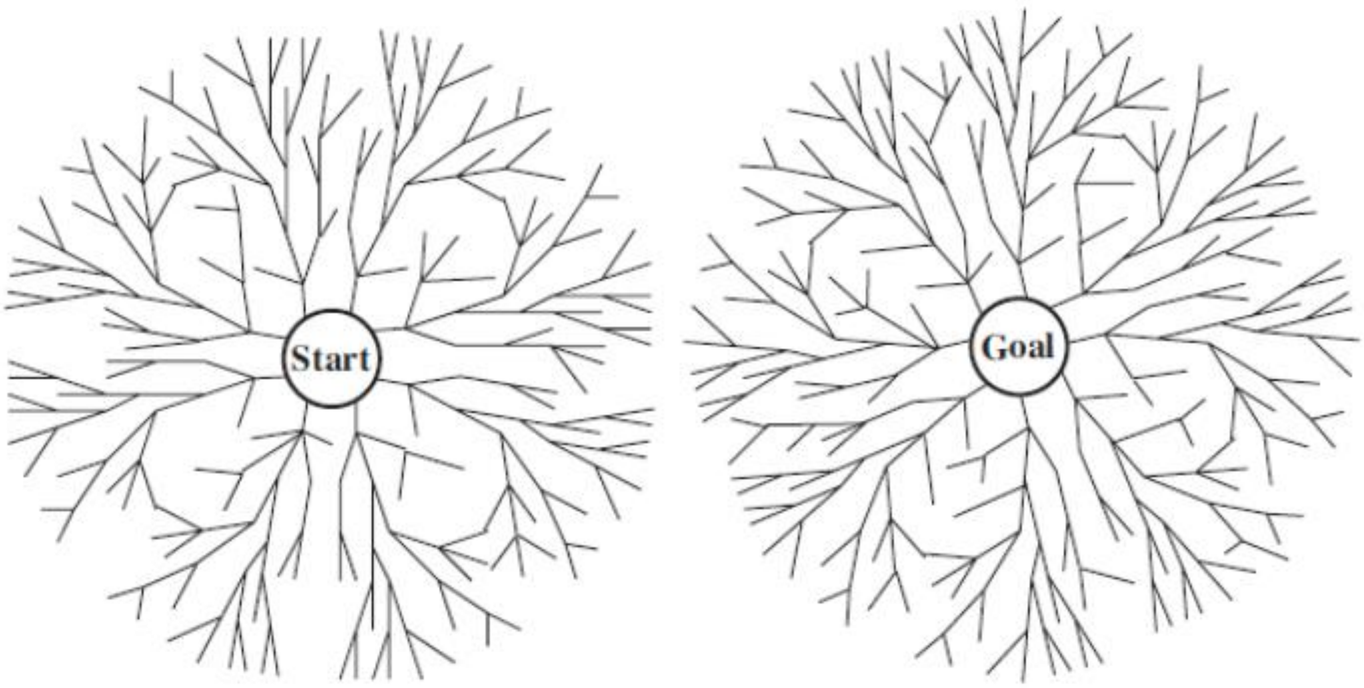


Iterative Deepening Search

- Evaluation:
  - Completeness: YES when the branching factor is finite
  - Time complexity: O(b^d)
  - Space complexity: O(bd)
  - Optimality: YES if costs are uniform

Bidirectional Search

- Two simultaneous searches: one from the initial state towards, and the other from the goal state backwards
- Hoping that two searches meet in the middle



Summary

- Comparison of uninformed algorithms (tree-search versions)

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Complete?	Yes^a	Yes^{a,b}	No	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}

Figure 3.21 Evaluation of tree-search strategies. b is the branching factor; d is the depth of the shallowest solution; m is the maximum depth of the search tree; l is the depth limit. Superscript caveats are as follows: ^{a} complete if b is finite; ^{b} complete if step costs $\geq \epsilon$ for positive ϵ; ^{c} optimal if step costs are all identical; ^{d} if both directions use breadth-first search.

Homework

- Conduct homework in the given notebook.

References

- Stuart Russell and Peter Norvig. 2009. Artificial Intelligence: A Modern Approach (3rd ed.). Prentice Hall Press, Upper Saddle River, NJ, USA.
- Lê Hoài Bắc, Tô Hoài Việt. 2014. Giáo trình Cơ sở Trí tuệ nhân tạo. Khoa Công nghệ Thông tin. Trường ĐH Khoa học Tự nhiên, ĐHQG-HCM.
- Nguyễn Ngọc Thảo, Nguyễn Hải Minh. 2020. Bài giảng Cơ sở Trí tuệ Nhân tạo. Khoa Công nghệ Thông tin. Trường ĐH Khoa học Tự nhiên, ĐHQG-HCM.