Pattern Matcher and Problem Solving with Searching

Keqiu Hu

(Based on Prof. Stolfo's lecture)

Outline

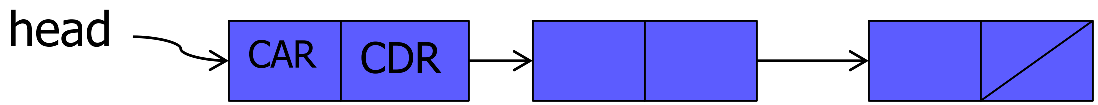
- Review Lisp
- Project One Supplement and some program snippet for project one(gifts)
- Problem Solving by searching

Lisp Review

- List functions
- Type determination functions
- Sequential control functions
- Some others .. for Project One
- Side effect in Setf

What is a List

- Primary data object
- Implemented as singly linked list:



- How to create a list?

(list 1 2 3) =>(1 2 3)

‘(a b c) => (A B C)

List Functions

- Car
  - Get first element in a list
    - Example: car ‘(a b) => a
- Cdr
  - Get the rest of the list
    - Example: cdr ‘(a b) => ‘(b)
- Cons
  - Push an element into a list
    - Example: cons ‘a ‘(b) => ‘(a b)
      cons ‘(a) ‘(b) => ‘(‘(a) b)

How to tell whether a variable is an Atom or List in LISP?

What is an ATOM?
=>Something doesn’t support the functions on the left..

Type Determination Functions

- Numberp
  - Is a number?
    - Example: numberp 1 => T
- Atom
  - Is an atom?
    - Example: atom 1 => T
- Listp
  - Is a list?
    - Example: listp ‘(a) => T
- Null
  - Is nil?
    - Example: null nil => T

Sequential Control Functions

- Cond
  - Similar to “Case” in C/C++/Java..
    - Example:

```
(cond
        (condition1 statement1)
        (condition2 statement2)
        (T statement3)
)
```

```
Case condition 1: statement1;break;
Case condition 2: statement2;break;
default:statement3
```

- If
  - Similar to “if” in C/C++/Java..
    - Example: (if boolean statement1 statement2)
    - if(boolean){statement1}else{statement2}

Indentation is extremely important if you are not good at counting parentheses!

Sequential Control Functions

- Examples:
  - Most Basic one!
    - (defun which_number(N)

```
          (cond
                ((equal N 1) 'One)
                (T 'Others)
          )
    )
```

Sequential Control Functions

- Examples:
  - How to sum a linked-list?
    - In Java (using loop)
    - Suppose “head” is head of a linked list:
      - sum = 0
      - While(head!=null){sum +=head.value;head=head.next;}
    - What if we don’t have WHILE/FOR loop?
      - We have the recursion: sum(head) = head.value+sum(head.next);
      - int sum(LinkNode X){

        If(x==null) return 0;

        return x.value+sum(x.next);

        }

Sequential Control Functions

- Examples:
  - Functional language open a door for you to think most naturally!

```
(defun sum (L)
        (cond
                ((NULL L) 0)
                (T (+ (car L) (sum (cdr L))))
        )
)
```

Sequential Control Functions

- Gift:
  - How to REVERSE A LINKED LIST using recursion?

    (Occurred in 50% of interviews)

    Hint: reverse(x) = reverse(x.next).append(x)

Other Functions

- Test equality:
  - Eq & Equal:

    Example:

    (setf L ‘(a b))

    (setf M ‘(a b))

    (eq L M) => NIL

    (equal L M) => T

Different from python:

```
a =1
b = 1
>>a is b
>> True
>> a == b
>> True
```

Other Functions

- elt
  - get an element from a sequence
- symbol-name
  - returns the name of a symbol as a string

    Example:

    - (defun startswithp (x) (equal (elt (symbol-name x) 0) #\P))

      =>checks if a symbol name starts with the letter p

Side effect in Setf

- Setf's side effect in list manipulation
  - Example:

    ```
    (setf L '(a b c))
    (setf Y (cons ‘d L))
    (setf (cadr y) ‘e)
    ```

- What is Y?
  - Y => (d e b c)
- What is L?
  - L=>(e b c)

Project One Supplement

- Requirement:
  - Matches pattern with data!
  - Special Marks:
    - Question mark “?”: match anything
    - Kleene star “*”: match 0 or more elements
    - Variable mark “?x”:
      - Binding one variable to .. an atom or list or whatever..
    - Exclamation mark/Ampersand/Greater/Smaller:
      - Indicate the relation between data and the value bounded to the variable x.

Project One Supplement

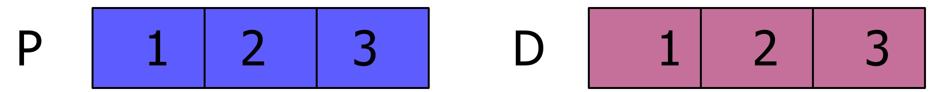
- Basic Examples:
  - match ‘(a) ‘(a)
    - Return T.
  - match ‘(1 2) ‘(2 1)
    - Return NIL.
  - match ‘(?x) ‘(4)
    - Return ((?x 4))
  - match ‘(? 7) ‘((6) 7)
    - Return T

Project One Supplement

- Examples:
  - match (?x) ()
    - Notice that () != (NIL)
    - Variable has to be bounded => return NIL
  - match (?x 2 ?x 4) (1 2 3 4)
    - Return NIL since ?x can not be both 1 and 3.
  - match (?x 5 6 ?y) (4 5 6 7)
    - Return (((?x 4)(?y 7))) Order not matter
  - match (* ?x * 7) (4 5 6 7)
    - Return (((?x 4))((?x 5))((?x 6))) Order not matter

First Gift for Project One

- I don't care symbol– The question mark "?"
  - How to match "?"
  - Think RECURSIVELY!



    - Suppose we have a pattern P and an input D

      (equal (car d) (car p))

      (match (cdr p)(cdr d))
    - Base case?
    - ((and (null P) (null D)) T)
    - Otherwise, if one NIL the other not
    - ((or (null P) (null D)) NIL)

First Gift for Project One

- I don't care symbol– The question mark "?"
  - (defun match (p d)

```
(cond
        ((and (null p) (null d)) T)
        ((or (null p) (null d)) NIL)
        ((or (equal (car p) '?) (equal (car d) (car p)))
                (match (cdr p)(cdr d))
         (T NIL)
)
)
```

First Gift for Project One

- I don't care symbol– The question mark "?"
  - Example:

    (match '(A ? C) '(A B C))

    =>return (match '(? C) '(B C)

    =>return (match '(C) '(C))

    =>return (match NIL NIL)

    =>return T

  - **(defun match (p d)**

```
(defun match (p d)
  (cond
    ((and (null p) (null d)) T)
    ((or (null p) (null d)) NIL)
    ((or (equal (car p) '?) (equal (car d) (car p)))
        (match (cdr p)(cdr d))
    (T NIL)
  )
)
```

Second Gift for Project One

- I don’t care how many symbol– The Kleene Star mark “*”
  - Previous Question Mark Matcher

```
(defun match (p d)
(cond
        ((and (null p) (null d)) T)
        ((or (null p) (null d)) NIL)
        ((or (equal (car p) '?) (equal (car d) (car p)))
                (match (cdr p)(cdr d))
        (T NIL)
        )
)
```

Second Gift for Project One

- I don't care how many symbol– The Kleene Star mark "*"
  - Previous Question Mark Matcher

```
(defun match (p d)
(cond
        ((and (null p) (null d)) T)
        ((or (null p) (null d)) NIL)
        ((or (equal (car p) '?) (equal (car d) (car p)))
              (match (cdr p)(cdr d))
        ((equal (car p) '*)  (or (match p (cdr d)) (match (cdr p) d)))
        (T NIL)
        )
)
```

Second Gift for Project One

- I don't care how many symbol– The Kleene Star mark "*"
  - Example:

(match '(A * C) '(A B C))

=>return (match '(* C) '(B C)

=>return

~~(match '(* C) C)~~

=> ~~(match '(* C) NIL~~)

or (match C C)

=>return T

```
(defun match (p d)
(cond
        ((and (null p) (null d)) T)
        ((or (null p) (null d)) NIL)
        ((or (equal (car p) '?) (equal (car d) (car p)))
                (match (cdr p)(cdr d)))
        ((equal (car p) '*)  (or (match p (cdr d)) (match (cdr
    p) d)))

        (T NIL)
        )
)
```

Second Gift for Project One

- Another Example:

  (match '(A * C) '(A C))

  =>return (match '(* C) '(C)

  =>return

  ~~(match '(* C) NIL)~~

  ~~=>return NIL~~

  or (match C C)

  =>return T

```
(defun match (p d)
(cond
        ((and (null p) (null d)) T)
        ((or (null p) (null d)) NIL)
        ((or (equal (car p) '?) (equal (car d) (car p)))
                (match (cdr p)(cdr d))
        ((equal (car p) '*)  (or (match p (cdr d)) (match (cdr
    p) d)))

        (T NIL)
        )
)
```

Third Gift for Project One

- Handle variables
  - Store variable bindings in a list
  - Check list if further matches appears.
  - Notice that variable can match ATOM/LIST!

Final Gift for Project One

- A horrible example:
  - match '((((((* ?x * (* ((* ?y * ?) (?z b)) (* ?u)) ? g *)))) * ?v ? t) '((((((8 x (z ((y x z f g) (z b)) (a b u)) g g)))) v t t)
  - =>
  - (((?V V) (?U U) (?Z Z) (?Y F) (?X X)) ((?V V) (?U B) (?Z Z) (?Y F) (?X X)) ((?V V) (?U U) (?Z Z) (?Y Z) (?X X)) ((?V V) (?U B) (?Z Z) (?Y Z) (?X X)) ((?V V) (?U U) (?Z Z) (?Y X) (?X X)) ((?V V) (?U B) (?Z Z) (?Y X) (?X X)) ((?V V) (?U U) (?Z Z) (?Y Y) (?X X)) ((?V V) (?U B) (?Z Z) (?Y Y) (?X X)) ((?V V) (?U U) (?Z Z) (?Y F) (?X 8)) ((?V V) (?U B) (?Z Z) (?Y F) (?X 8)) ((?V V) (?U U) (?Z Z) (?Y Z) (?X 8)) ((?V V) (?U B) (?Z Z) (?Y Z) (?X 8)) ((?V V) (?U U) (?Z Z) (?Y X) (?X 8)) ((?V V) (?U B) (?Z Z) (?Y X) (?X 8)) ((?V V) (?U U) (?Z Z) (?Y Y) (?X 8)) ((?V V) (?U B) (?Z Z) (?Y Y) (?X 8)))

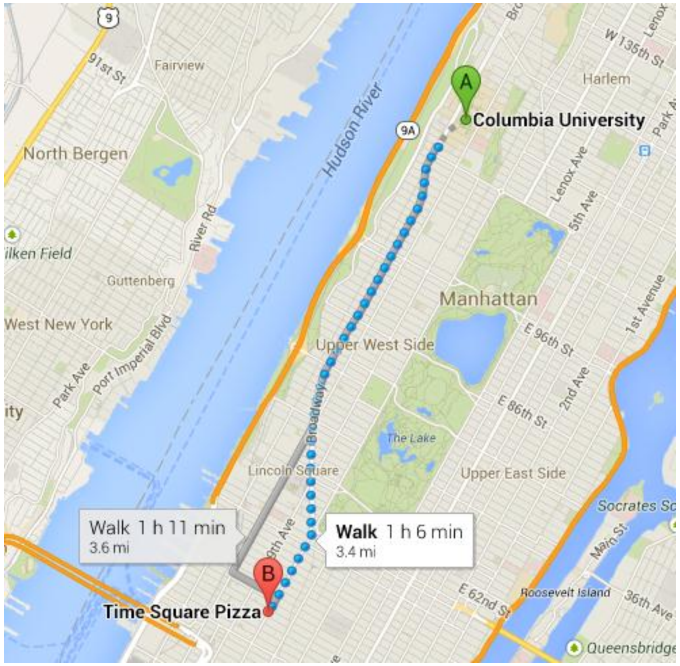
Solving problems by searching

Chapter 3

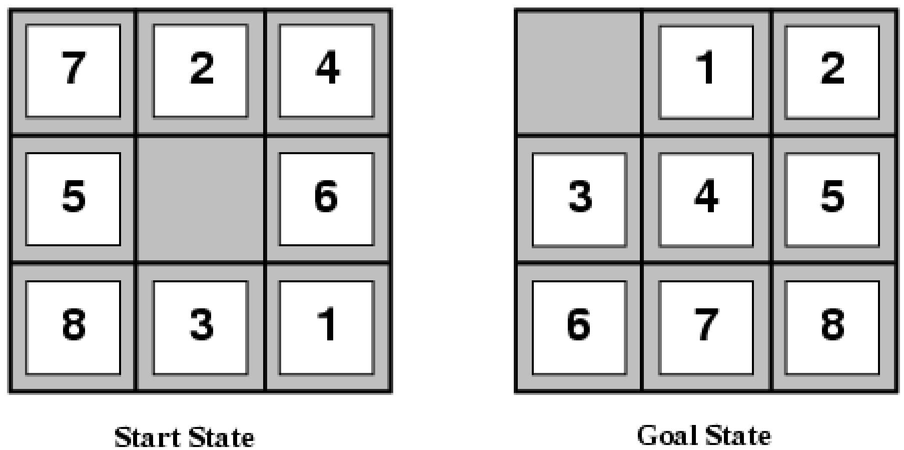
Problem Solving

State – Operator – Search

- A problem e.g., "How can I get to Time Square?"
  - I. A state -> A configuration of current environment
    - Eg: What is the position of me?
    - Columbia University :
      Latitude - Longitude:
      40.806963,-73.961624
  - II. An operator -> Function maps state to state
    - Eg: How should I move?
    - Move NORTH/SOUTH/WEST/EAST?
  - III. Initial State and Goal State -> Seek a sequence of operators
    Columbia University :
    Latitude - Longitude:
    (40.806963,-73.961624)
    Time Square:
    Latitude - Longitude:
    (27.813054,-80.425241)

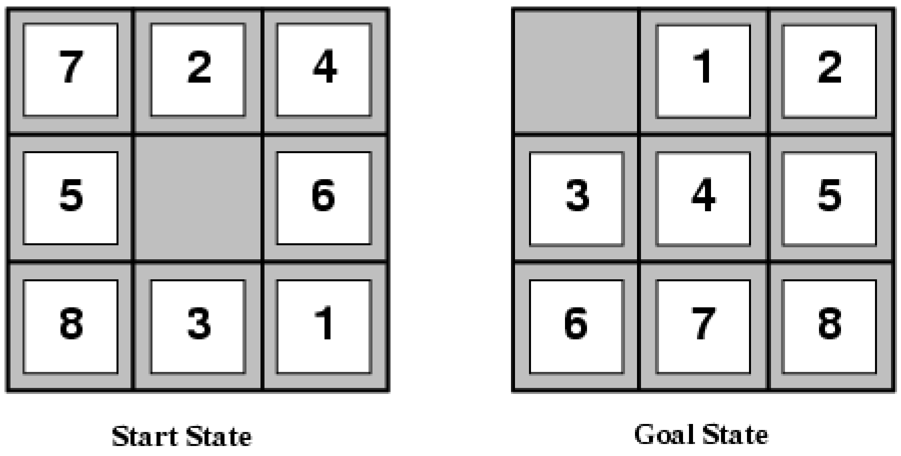


Example: The 8-puzzle



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

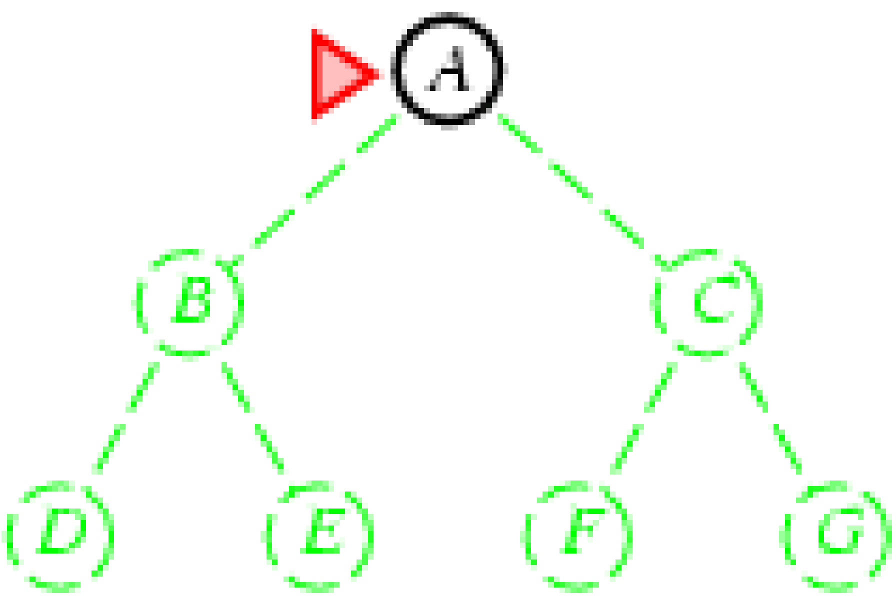
Example: The 8-puzzle



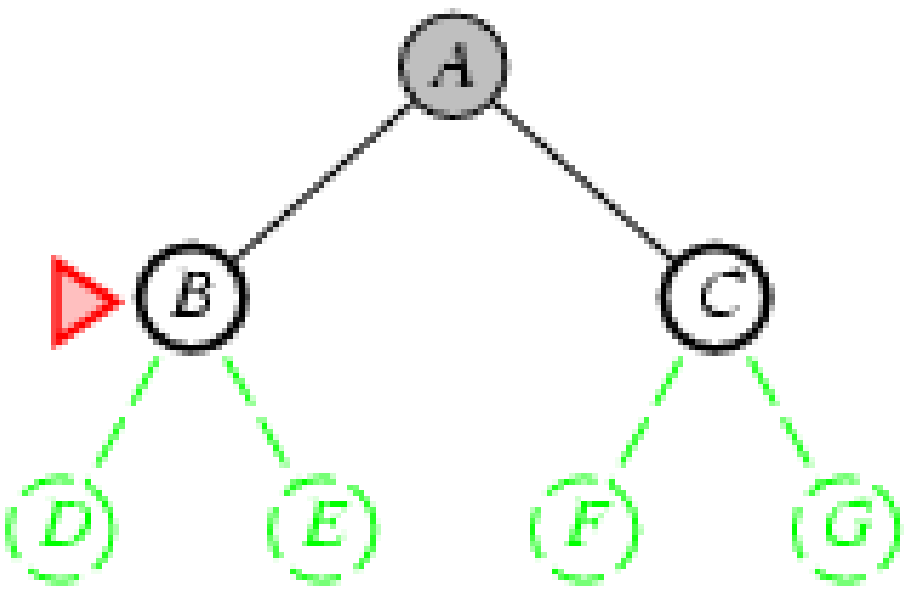
- states? locations of tiles
- actions? move blank left, right, up, down
- goal test? = goal state (given)
- path cost? 1 per move
-

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

Breadth-first search

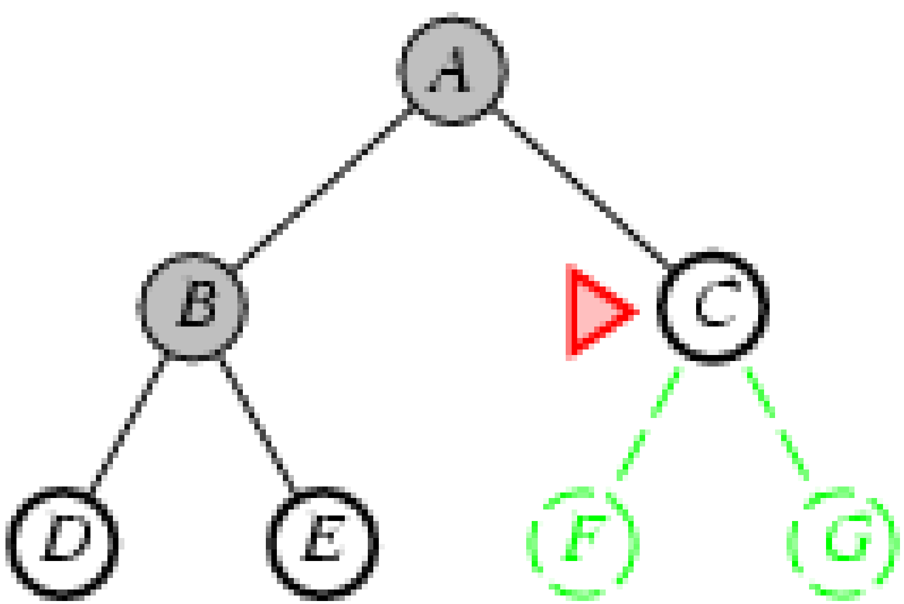
- Expand shallowest unexpanded node
- Implementation:
  - *fringe* is a FIFO queue, i.e., new successors go at end
  - 

Breadth-first search

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

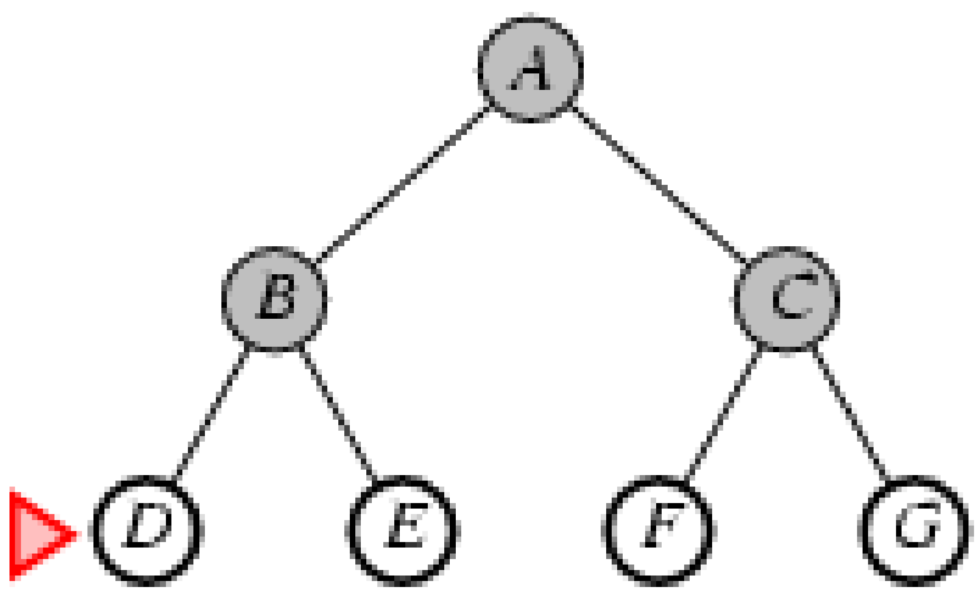
Breadth-first search

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



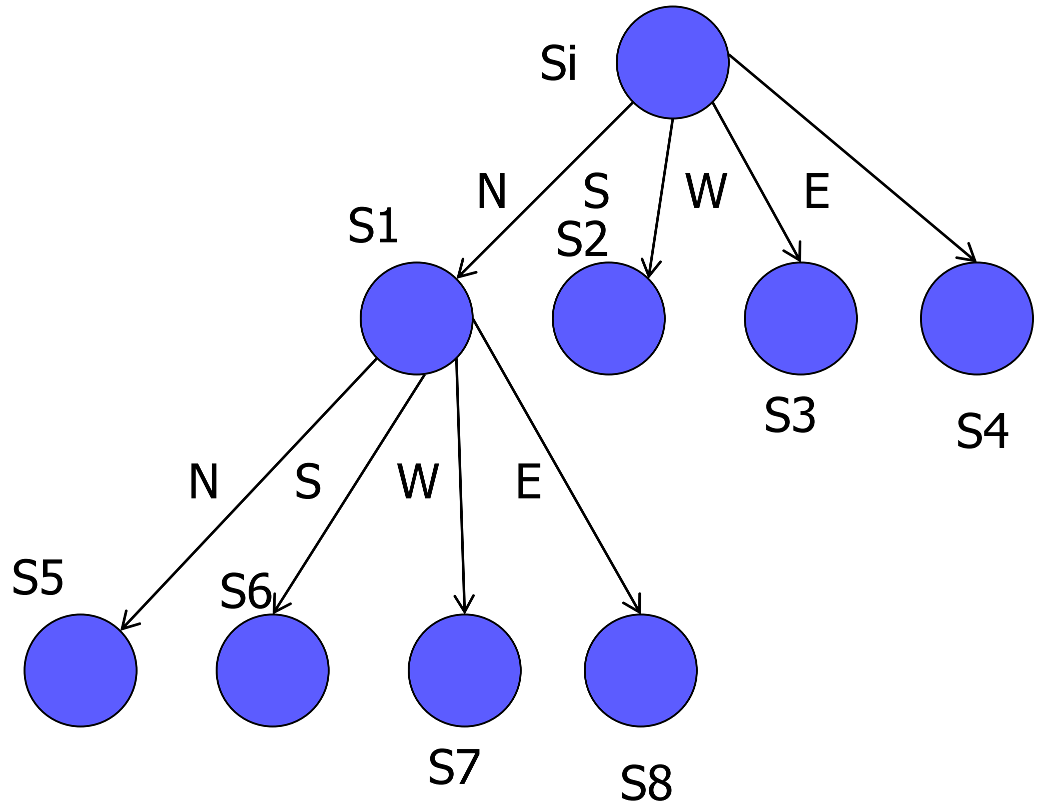
Breadth-first search

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



Breadth-first search

- Pruning
  - Operator: North/South/West/East



Breadth-first search

- Strategy
  - Put Si on OPEN list
  - If OPEN is empty exit with FAIL
  - Remove first item from OPEN, call it N
    - [Add N to CLOSE list]
    - If N == Goal State, exit with SUCCESS
  - Add all nodes in Successor(N) that IS NOT in CLOSED list to OPEN
    - Continue..

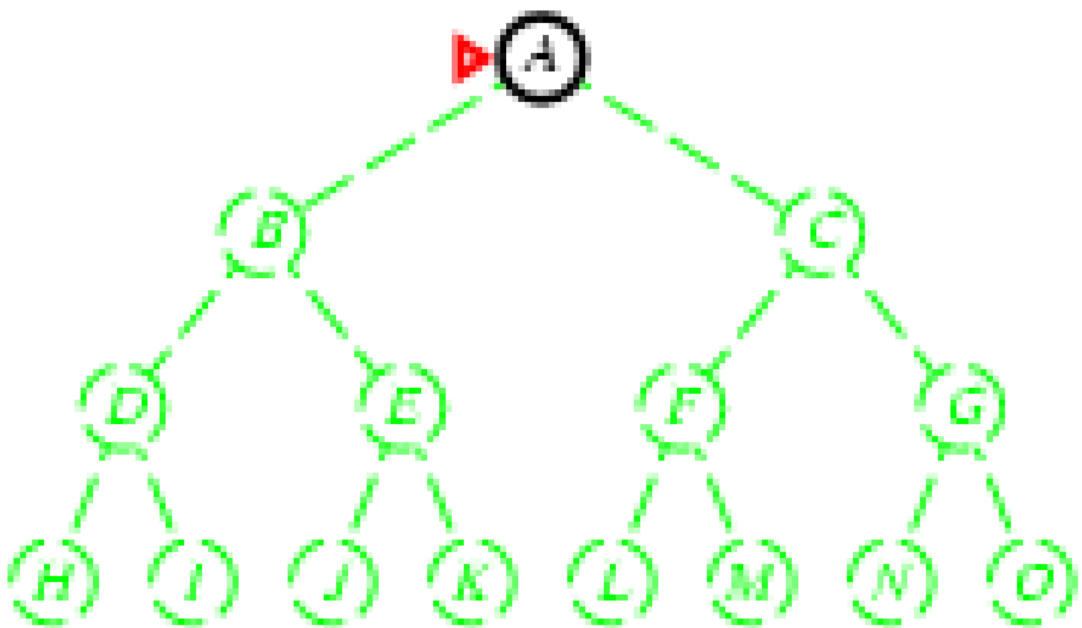
Properties of breadth-first search

- 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)

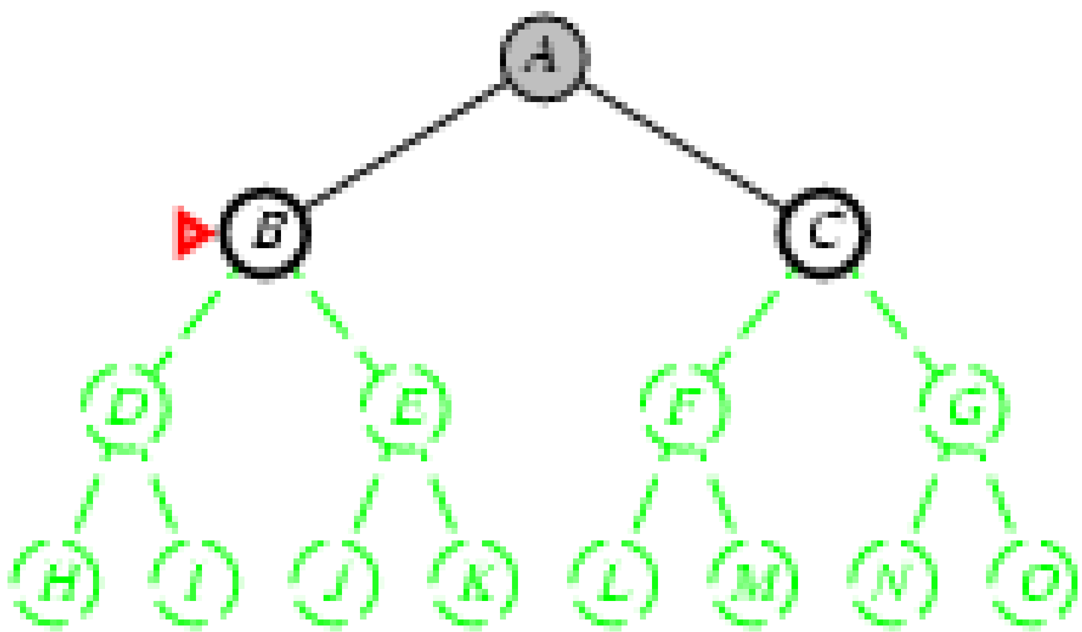
Depth-first search

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



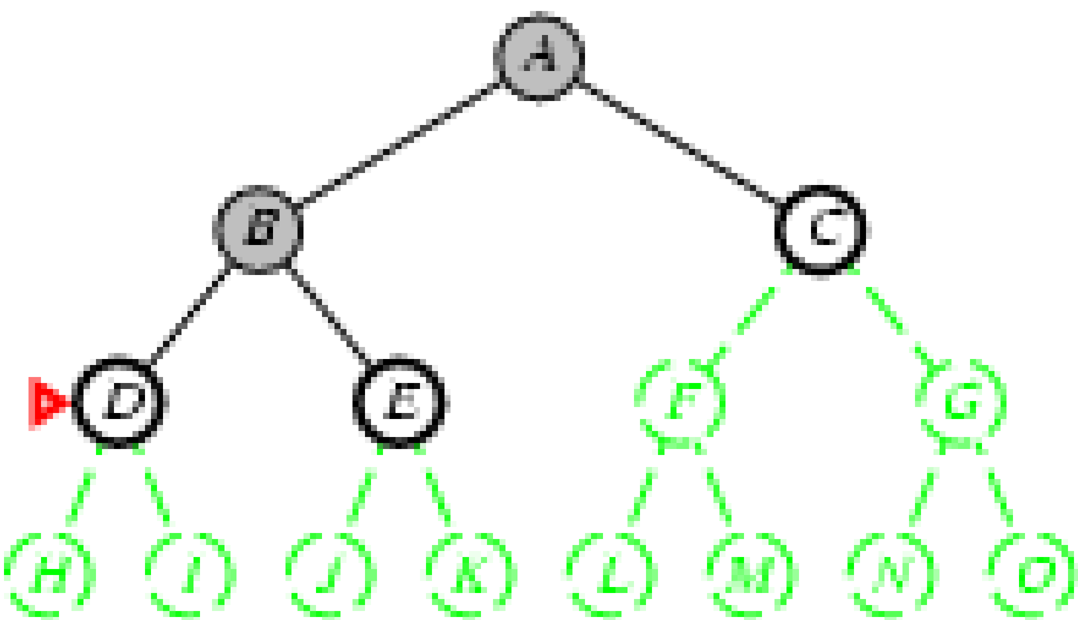
Depth-first search

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



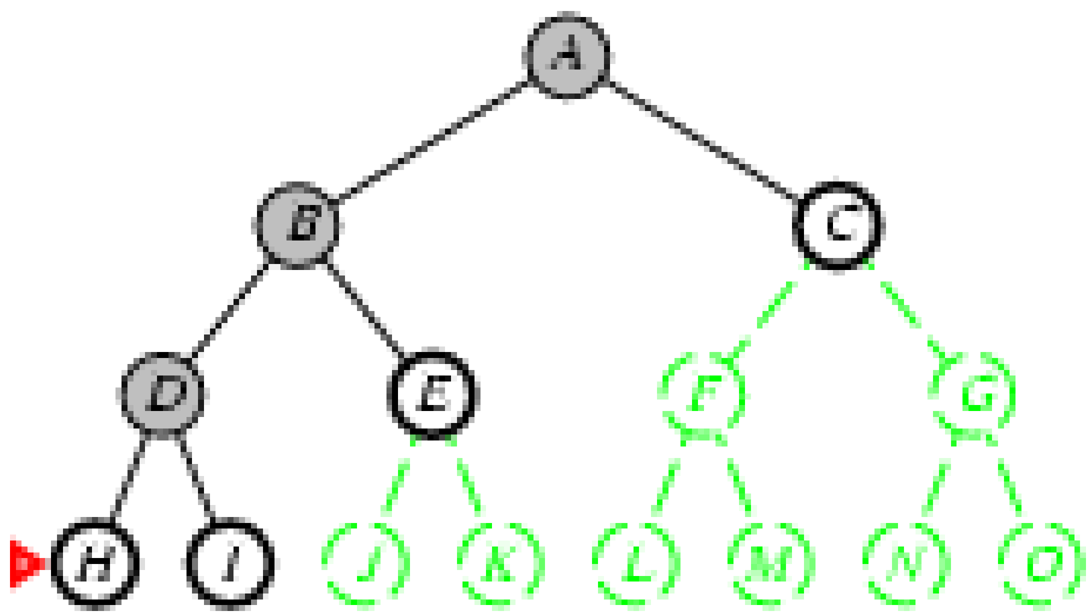
Depth-first search

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



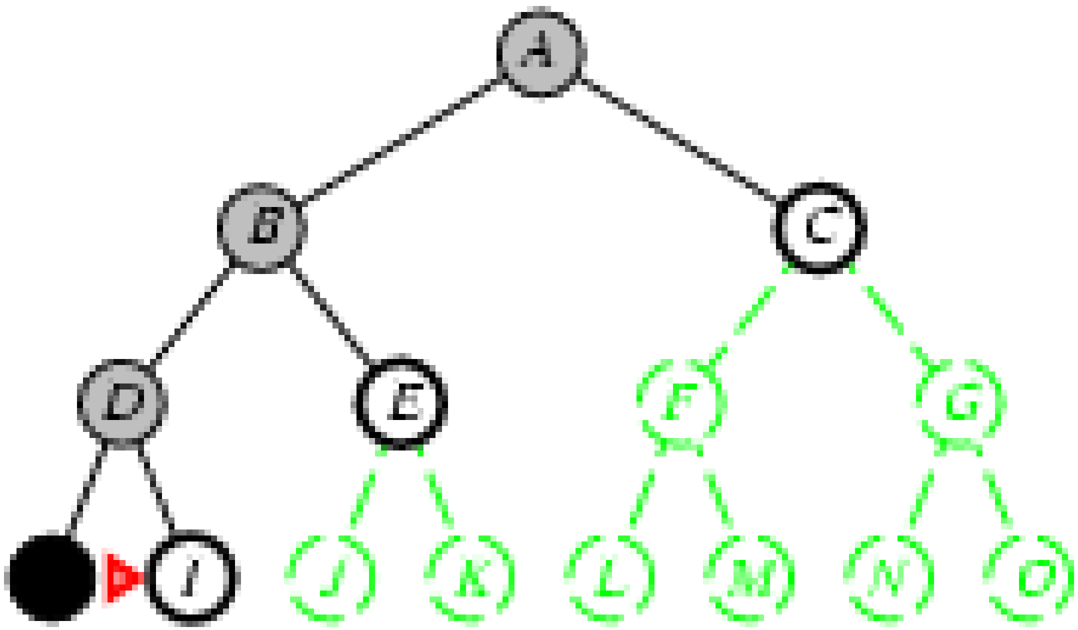
Depth-first search

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



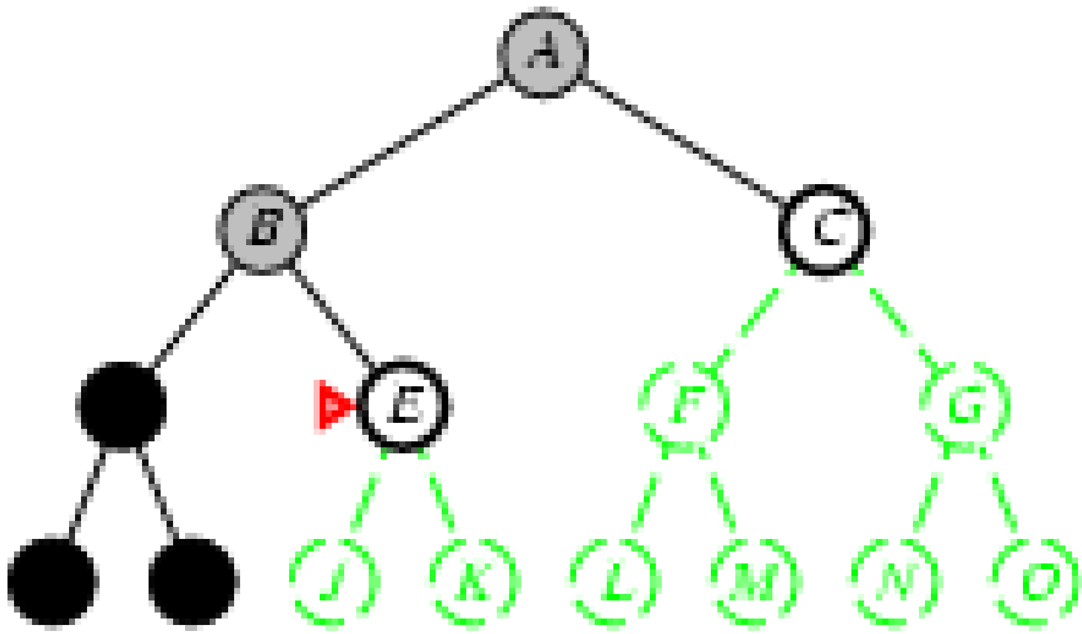
Depth-first search

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



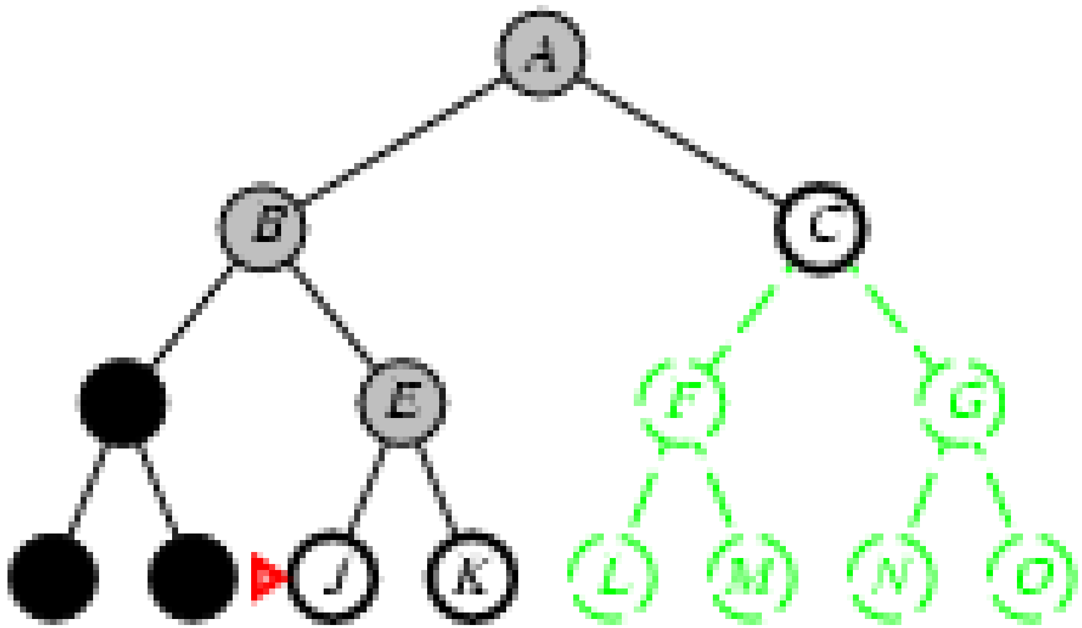
Depth-first search

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



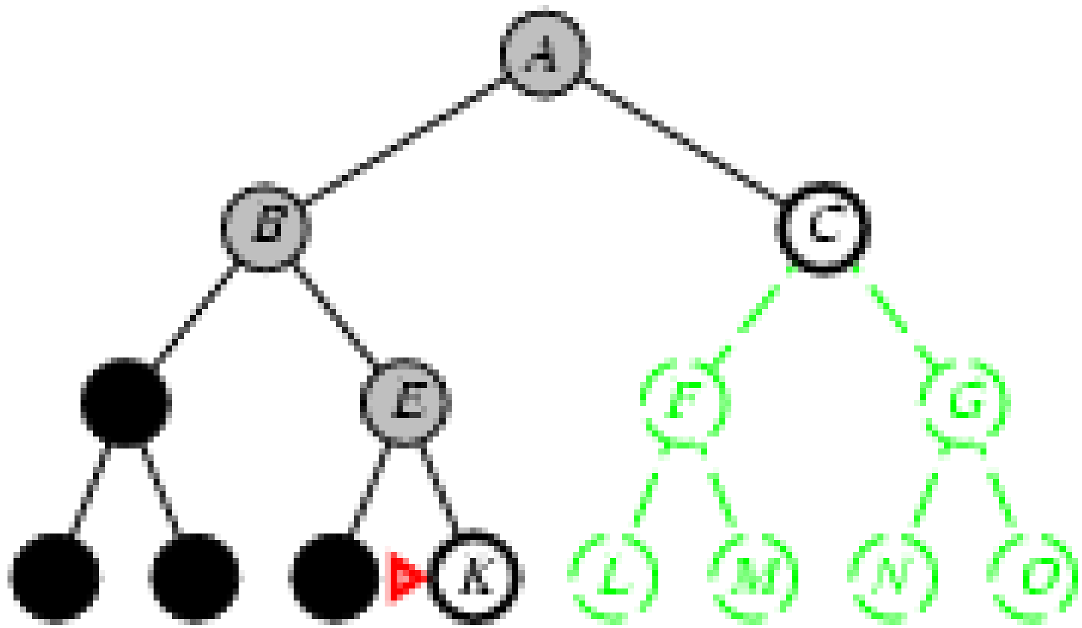
Depth-first search

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



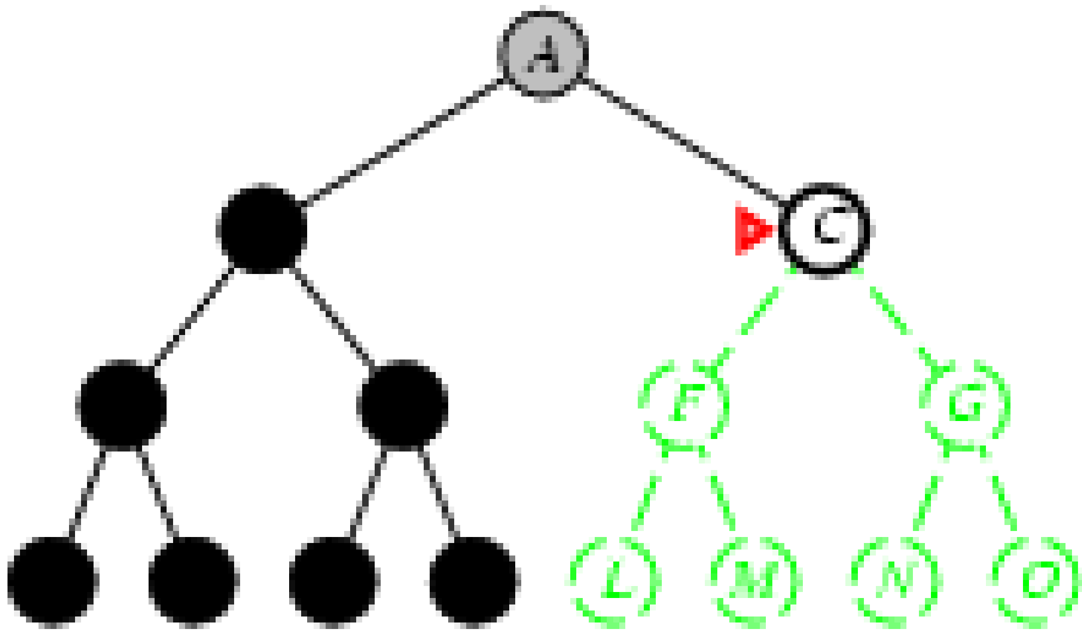
Depth-first search

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



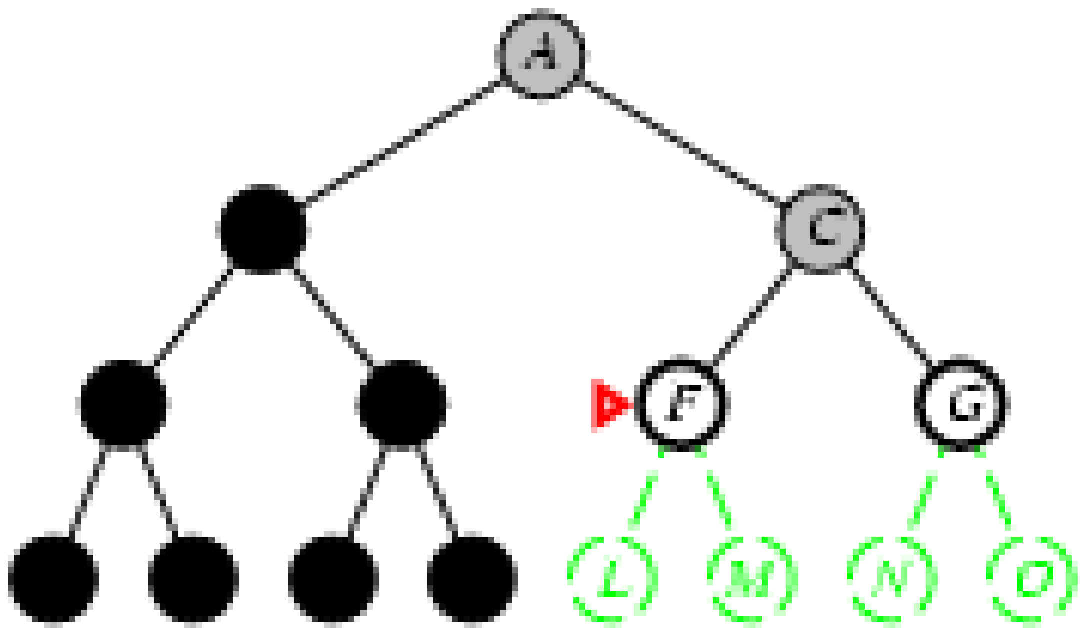
Depth-first search

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



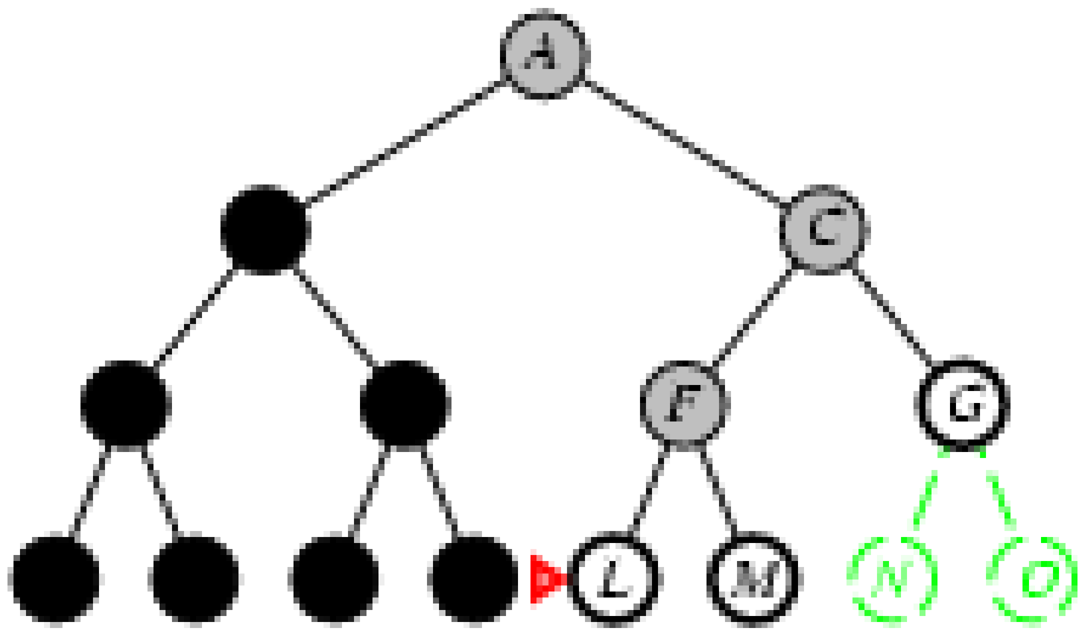
Depth-first search

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



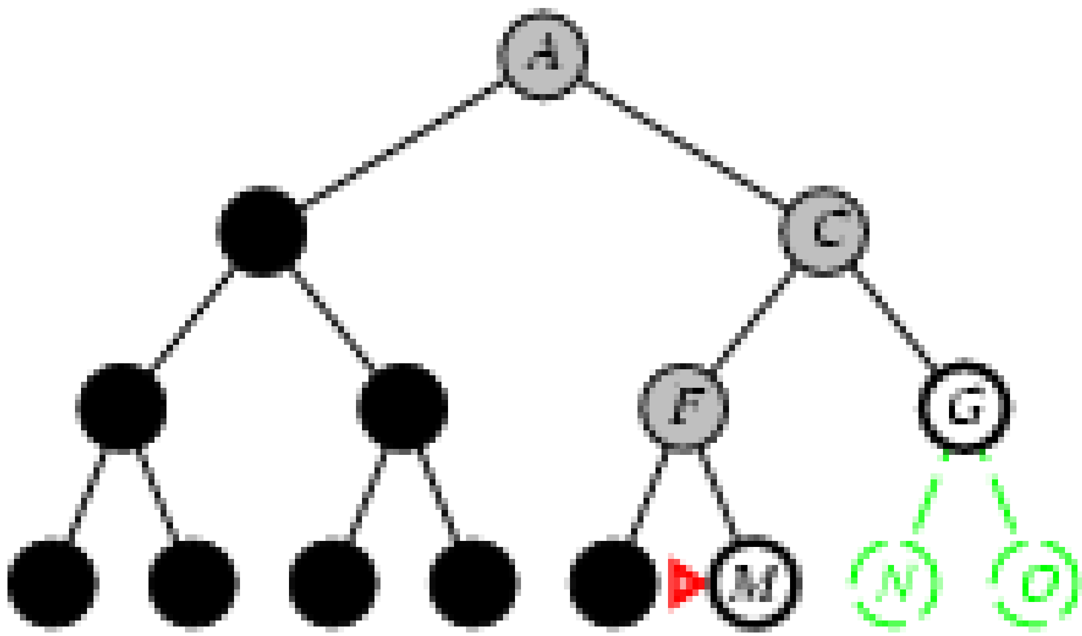
Depth-first search

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



Depth-first search

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



Properties of depth-first search

- Complete? No: fails in infinite-depth spaces, spaces with loops
  - Modify to avoid repeated states along path
    → complete in finite spaces
- Time? $O(b^m)$: terrible if m is much larger than d
  - but if solutions are dense, may be much faster than breadth-first
- Space? $O(bm)$, i.e., linear space!
- Optimal? No

Depth-limited search

= depth-first search with depth limit *l*,

i.e., nodes at depth *l* have no successors

- Recursive implementation:

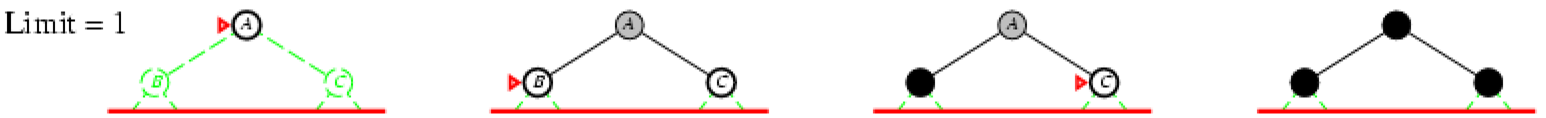
```
function DEPTH-LIMITED-SEARCH( problem, limit) returns soln/fail/cutoff
    RECURSIVE-DLS(MAKE-NODE(INITIAL-STATE[problem]), problem, limit)

function RECURSIVE-DLS(node, problem, limit) returns soln/fail/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

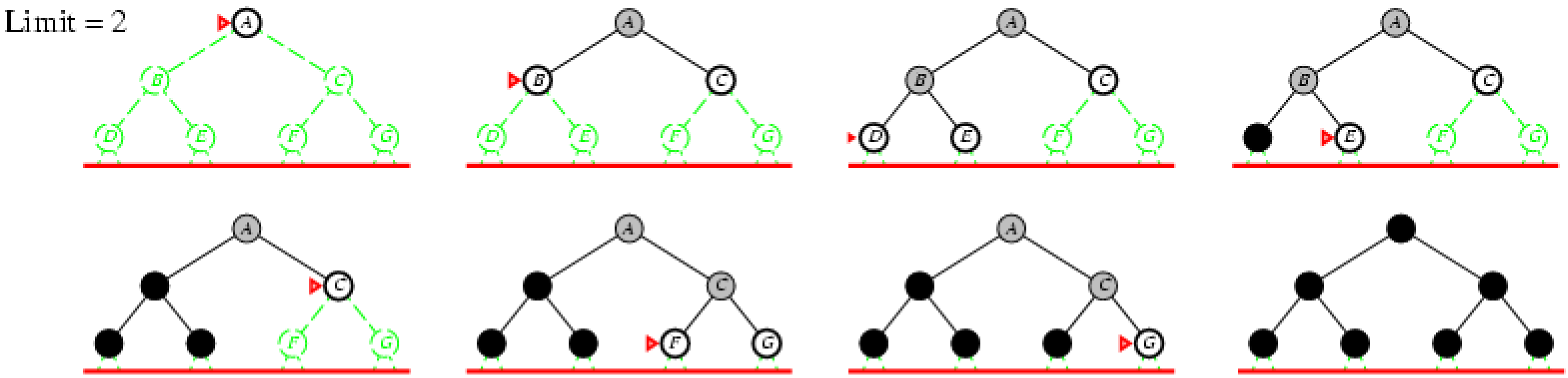
```
function ITERATIVE-DEEPENING-SEARCH(problem) returns a solution, or fail-
ure
   inputs: problem, a problem

   for depth ← 0 to ∞ do
      result ← DEPTH-LIMITED-SEARCH(problem, depth)
      if result ≠ cutoff then return result
```

Iterative deepening search *l* =0

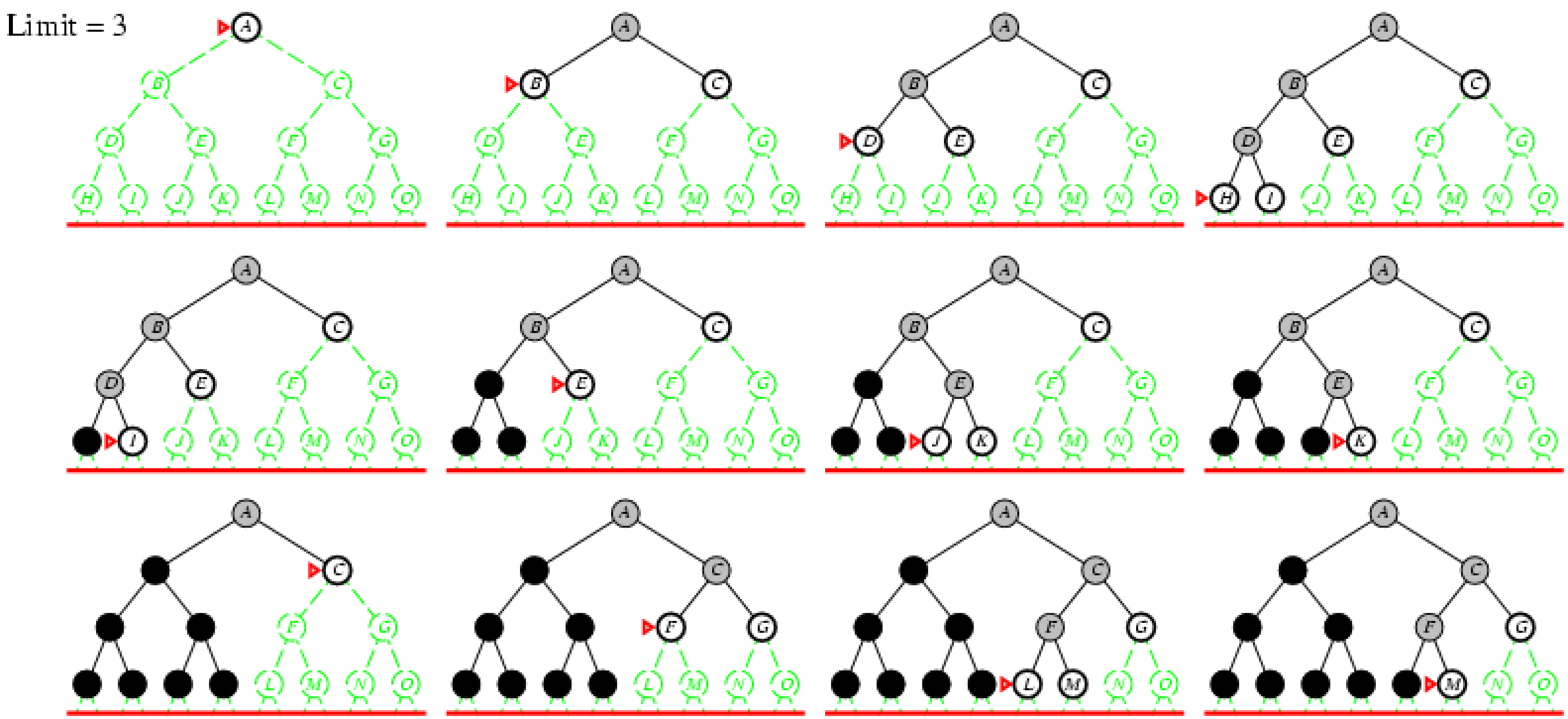
Iterative deepening search *l* =1



Iterative deepening search l =2



Iterative deepening search *l* =3



Iterative deepening search

- Number of nodes generated in a depth-limited search to depth *d* with branching factor *b*:

$$N_{DLS} = b^0 + b^1 + b^2 + \ldots + b^{d-2} + b^{d-1} + b^d$$

- Number of nodes generated in an iterative deepening search to depth *d* with branching factor *b*:

$$N_{IDS} = (d+1)b^0 + d\ b\text{^}^1 + (d-1)b\text{^}^2 + \ldots + 3b^{d-2} + 2b^{d-1} + 1b^d$$

- For *b = 10*, *d = 5*,
  - N_{DLS} = 1 + 10 + 100 + 1,000 + 10,000 + 100,000 = 111,111
  - N_{IDS} = 6 + 50 + 400 + 3,000 + 20,000 + 100,000 = 123,456

- Overhead = (123,456 - 111,111)/111,111 = 11%

Properties of iterative deepening search

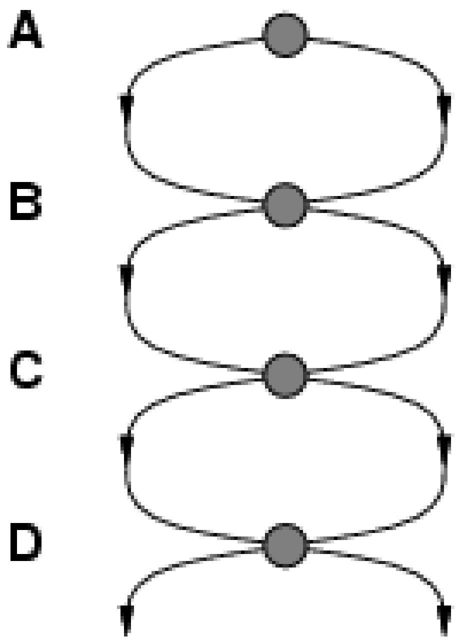
- Complete? Yes
- Time? $(d+1)b^0 + d\, b^1 + (d-1)b^2 + \ldots + b^d = O(b^d)$
- Space? $O(bd)$
- Optimal? Yes, if step cost = 1

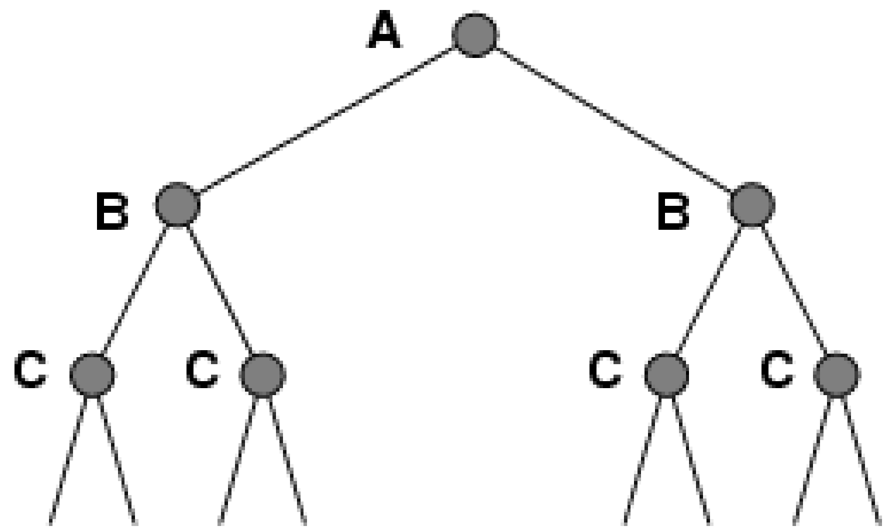
Summary of algorithms

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening
Complete?	Yes	Yes	No	No	Yes
Time	$O(b^{d+1})$	$O(b^{\lceil C^*/\epsilon \rceil})$	$O(b^m)$	$O(b^l)$	$O(b^d)$
Space	$O(b^{d+1})$	$O(b^{\lceil C^*/\epsilon \rceil})$	$O(bm)$	$O(bl)$	$O(bd)$
Optimal?	Yes	Yes	No	No	Yes

Repeated states

- Failure to detect repeated states can turn a linear problem into an exponential one!





Graph search

```
function GRAPH-SEARCH(problem, fringe) returns a solution, or failure
    closed ← an empty set
    fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
    loop do
        if fringe is empty then return failure
        node ← REMOVE-FRONT(fringe)
        if GOAL-TEST[problem](STATE[node]) then return SOLUTION(node)
        if STATE[node] is not in closed then
            add STATE[node] to closed
            fringe ← INSERTALL(EXPAND(node, problem), fringe)
```

Summary

- Problem formulation usually requires abstracting away real-world details to define a state space that can feasibly be explored
- Variety of uninformed search strategies
- Iterative deepening search uses only linear space and not much more time than other uninformed algorithms