RN, Chapter 3.4 – 3.6

Blind Search

Some material from: D Lin, J You, JC Latombe

Search Overview

- Introduction to Search
- Blind Search Techniques
  aka "Uninformed Search" (Goal vs NonGoal)
  - Breadth-First (Uniform Cost)
  - Depth-First
  - "Iterative Deepening"
  - Bi-Directional
- Heuristic Search Techniques
- Stochastic Algorithms
- Game Playing search
- Constraint Satisfaction Problems

Generic Search Algorithm

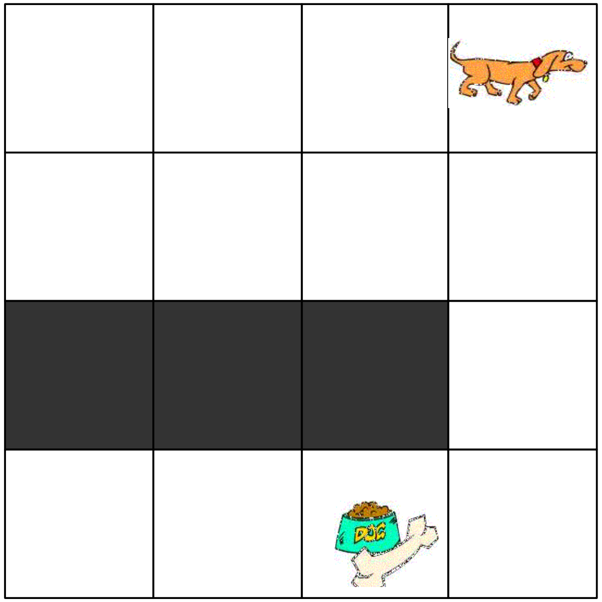
```
Search_insert( start, operations, isGoal ): path
  L = make-queue( start )
  loop
      n := pop( L )
      if [ isGoal( n )]
        return( n )
      S := successors( n, operators )
      L := insert( S, L )
  until L is empty
  return( failure )
```

insert could be queue, stack, . . .

defines strategy!

Blind Search

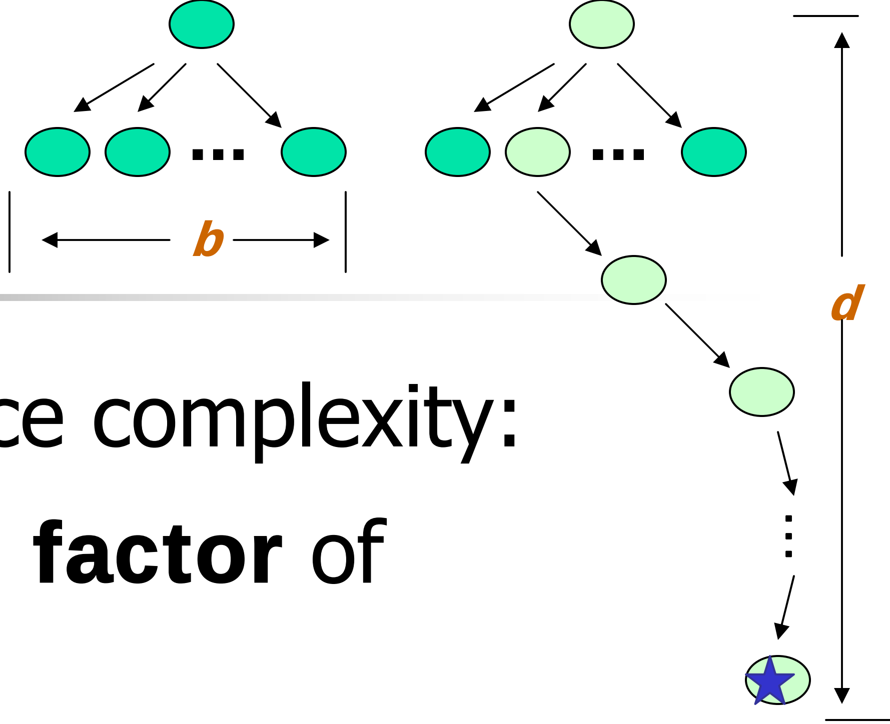
- Blind Search
  - Depth-first search
  - Breadth-first search
  - Iterative deepening
  - …
- Not "guided" by goal
- No matter where the goal is, these algorithms will do the same thing.



Performance Measures of Search Algorithms

- Completeness
  Does algorithm always find **a sol'n** (if ∃)?
- Optimality
  Does it always find **least cost** sol'n?
- Time complexity
  How **long does it take** to find sol'n?
- Space complexity
  How much **memory is required** to find a sol'n?

Parameters

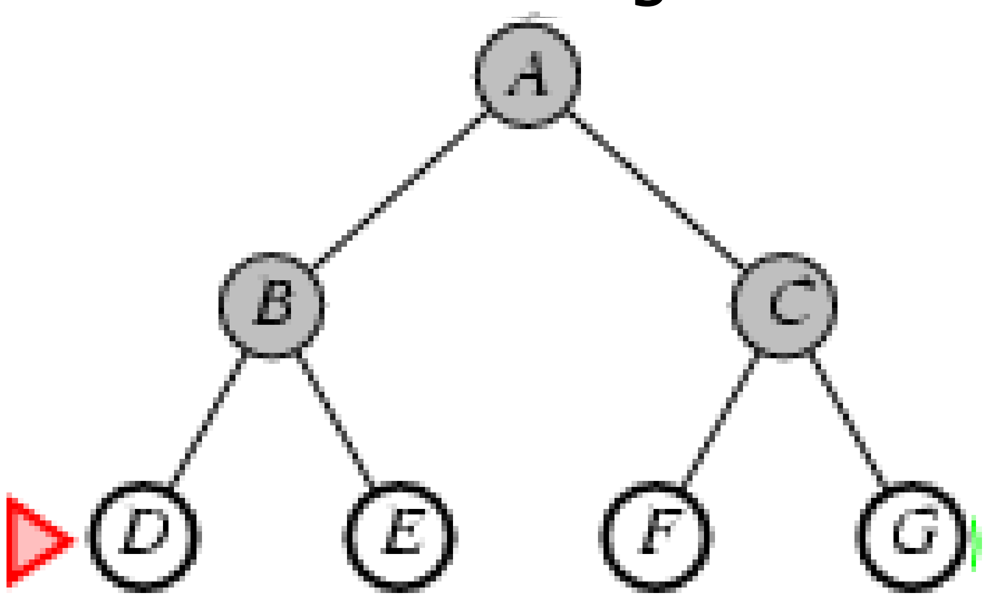


To measure Time and Space complexity:

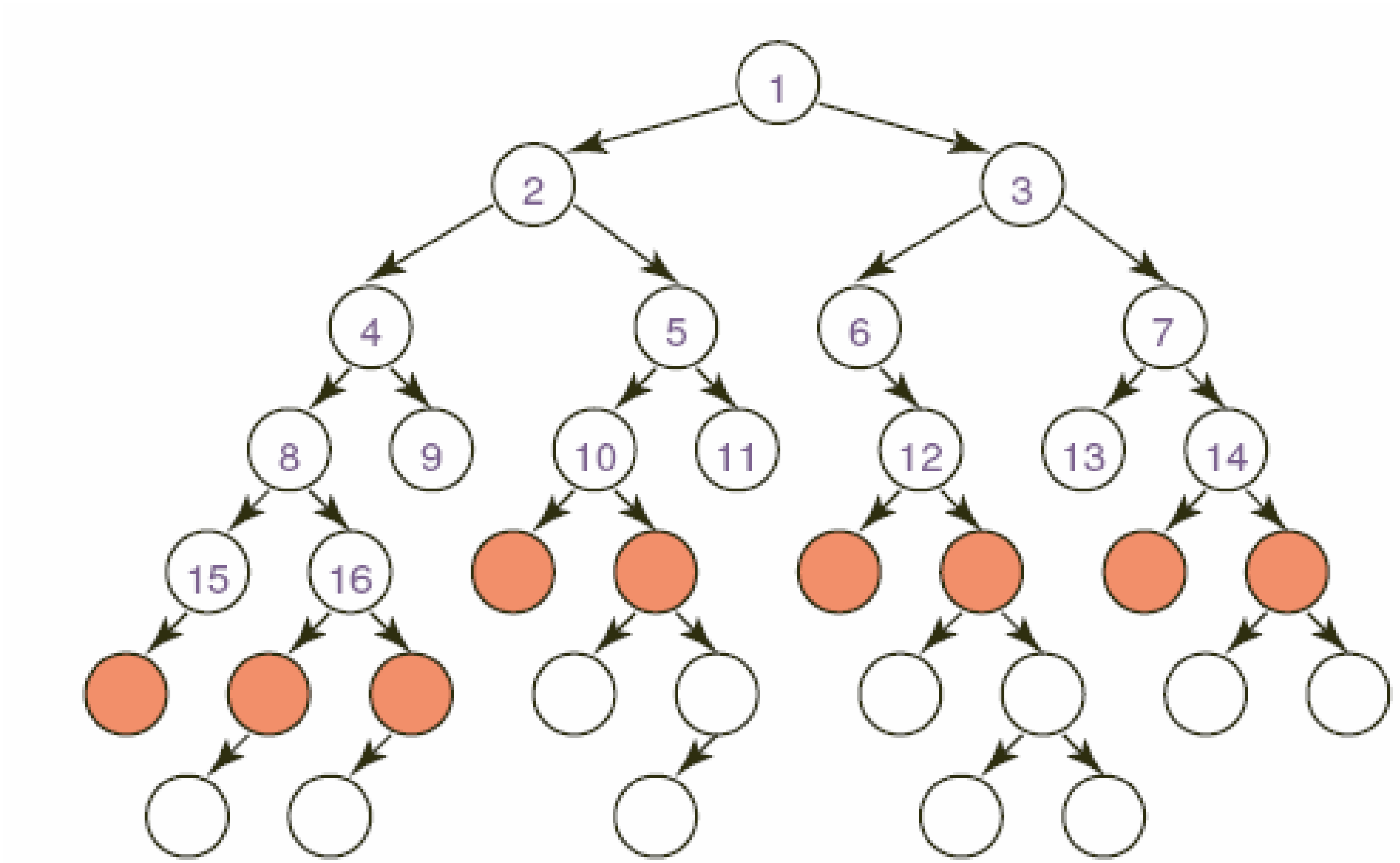
- ***b***: maximum **branching factor** of the search tree
  - Max number of operations at any state
- ***d***: **depth** of the least-cost solution
  - depth of shallowest goal node in search tree
- ***m***: maximum depth of the state space (may be ∞)

Breadth-first search

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



Breadth-First Search



Properties of Breadth-First search

- Complete? Yes (if *b* is finite)
- Optimal? Yes (if cost = 1 per step)
- Time? $O(b^d)$
  - $1 + b + b^2 + \ldots + b^d/2 = O(b^d)$
- Space? $O(b^d)$
  - keeps every intermediate node in memory
- Space is the bigger problem (more than time)

Time and Memory Requirements

d	#Nodes	Time	Memory
2	111	.01 msec	11 Kbytes
4	11,111	1 msec	1 Mbyte
6	$\sim 10^6$	1 sec	100 Mb
8	$\sim 10^8$	100 sec	10 Gbytes
10	$\sim 10^{10}$	2.8 hours	1 Tbyte
12	$\sim 10^{12}$	11.6 days	100 Tbytes
14	$\sim 10^{14}$	3.2 years	10,000 Tb

Assumptions: b = 10; 1,000,000 nodes/sec; 100bytes/node

Uniform Cost Search

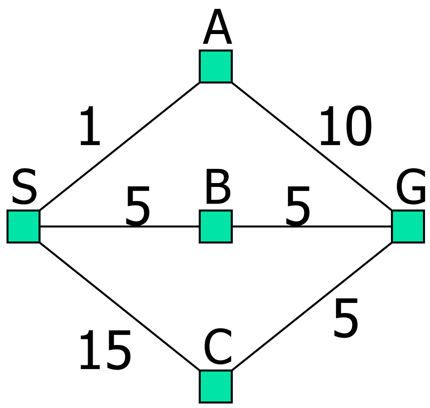
- BreadthFirst returns SHALLOW-est Goal
  … not necessarily best. . .
- Uniform Cost Search:

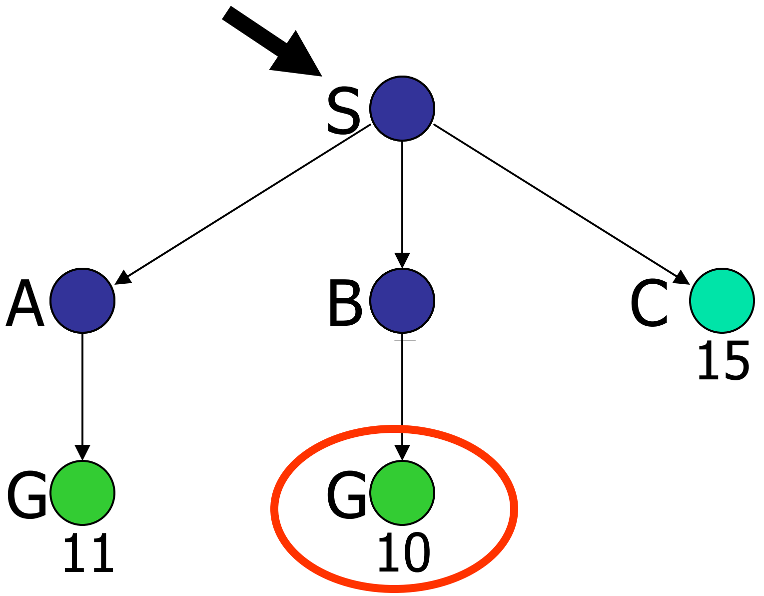
  Expand LEAST Cost node

- If COST ≡ Depth, then UC = BF

To insure optimality…

- To guarantee OPTIMAL path, need to maintain queue, sorted in increasing order:



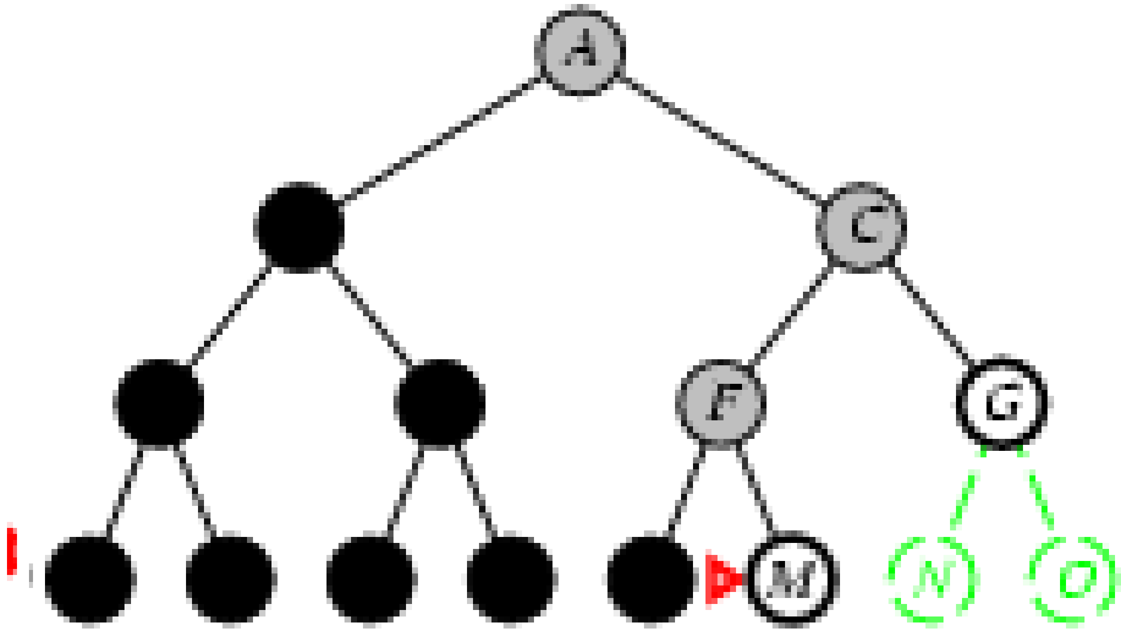


Uniform-cost search

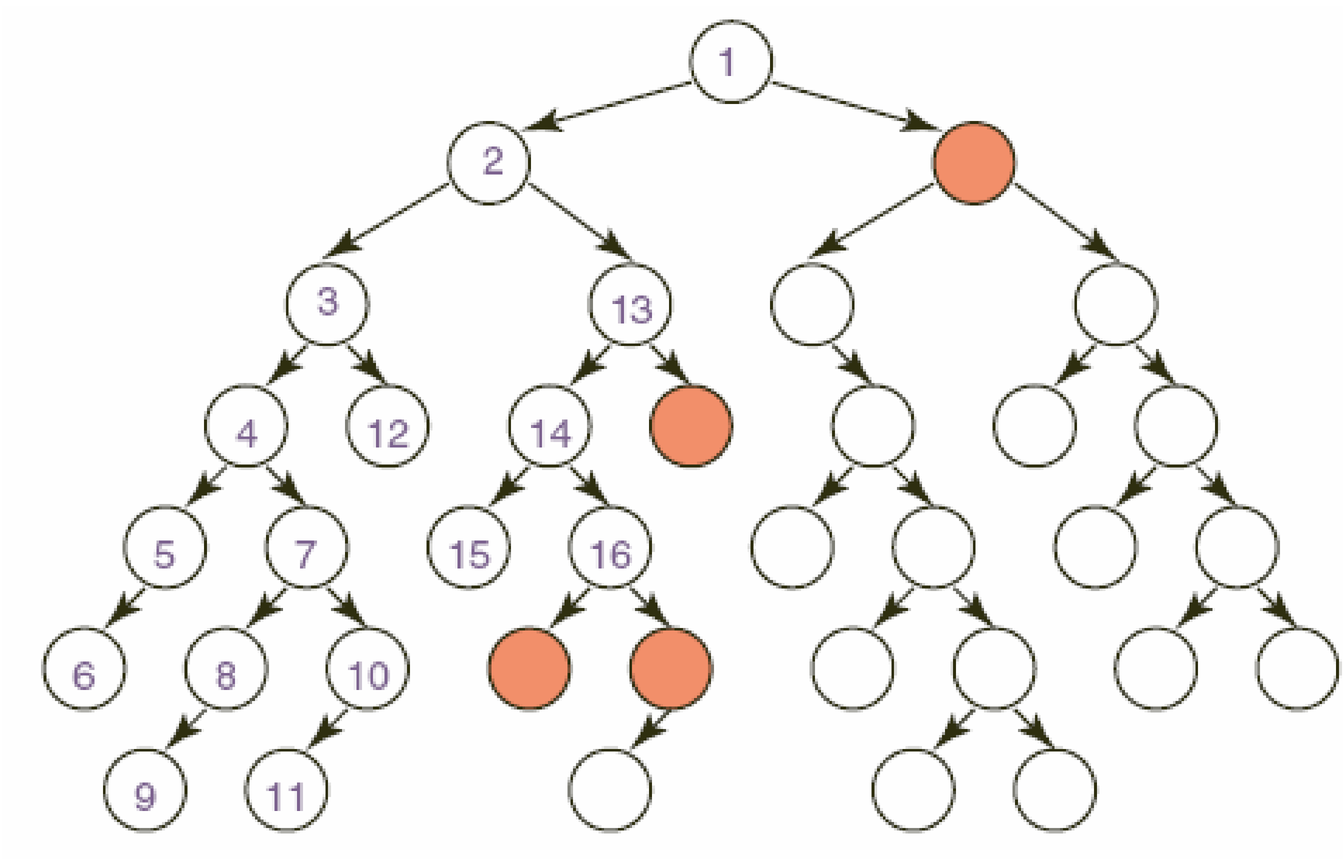
- Expand least-cost unexpanded node
- Implementation:
  - *fringe* = queue ordered by path cost
- Equivalent to breadth-first if step costs all equal
- Complete? *Yes*, if step cost ≥ ε
  (in trouble if cost = 0)
- Optimal? *Yes*
  …as nodes expanded in increasing order of cost
- Time? $O(b^{\lceil C^*/\varepsilon \rceil})$
  where C^* = cost of optimal solution
- Space? $O(b^{\lceil C^*/\varepsilon \rceil})$

Depth-first search

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



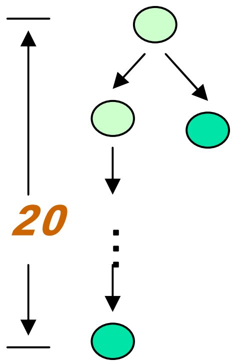
Depth-first Search



Properties of Depth-First Search

- <u>Complete?</u> **No**: fails in infinite-depth spaces, or if loops
  - Modify to avoid repeated states along path → complete in finite spaces
- <u>Optimal?</u> **No**

  … first found ≠? best



- <u>Time?</u> **$O(b^m)$:**
  - terrible if *m* is much larger than *d*
  - but if solutions are dense, may be much faster than BF
- <u>Space?</u> ***(b m)***
  - *d=12 ⇒ 12 kb, not 111 terabytes!*

BFS/UC vs. DFS

	Complete?	Optimal?	Time	Space
BFS/UC	YES	YES	b^d	b^d
DFS	finite depth	NO	b^m	b m

- Time:
  - m=d DFS typicall
- DFS almost always beats BFS

- Challenge:
  How to get BFS's guarantees, using only DFS's memory??

Which Strategy to Use?

- Depends on problem.
- If there are infinite paths

  $\Rightarrow$ depth-first is bad
- If goal is at known depth

  $\Rightarrow$ depth-first is good
- If $\exists$ large (possibly ∞) branching factor

  $\Rightarrow$ breadth-first is bad

(Could try nondeterministic search:
Expand an open node at random.)

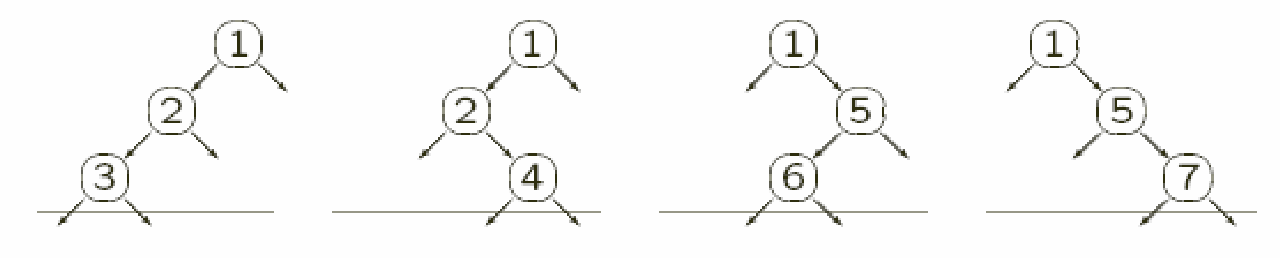
Depth-Limited Strategy

- Depth-first with depth cut-off k
  (do NOT expand nodes below depth k)

- Three possible outcomes:
  - Solution
  - Failure (no solution)
  - Cutoff (no solution within cutoff)

Depth-Limited Depth-First-Search

- Depth cut-off: $k=3$



- Complete: No unless soln @ depth $\leq k$
- Optimal: No
- Time: $O(b^k)$
- Space: $O(b\ k)$

Iterative Deepening Strategy

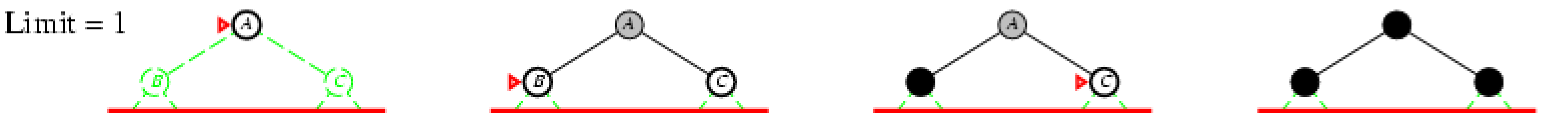
Use an artificial depth cutoff, *k*.

- For k = 1…
  - Use Depth-limited Depth-First Search(k)
  - If succeeds: DONE.
  - If not: increase *k* by 1
    (Regenerate nodes, as necessary)

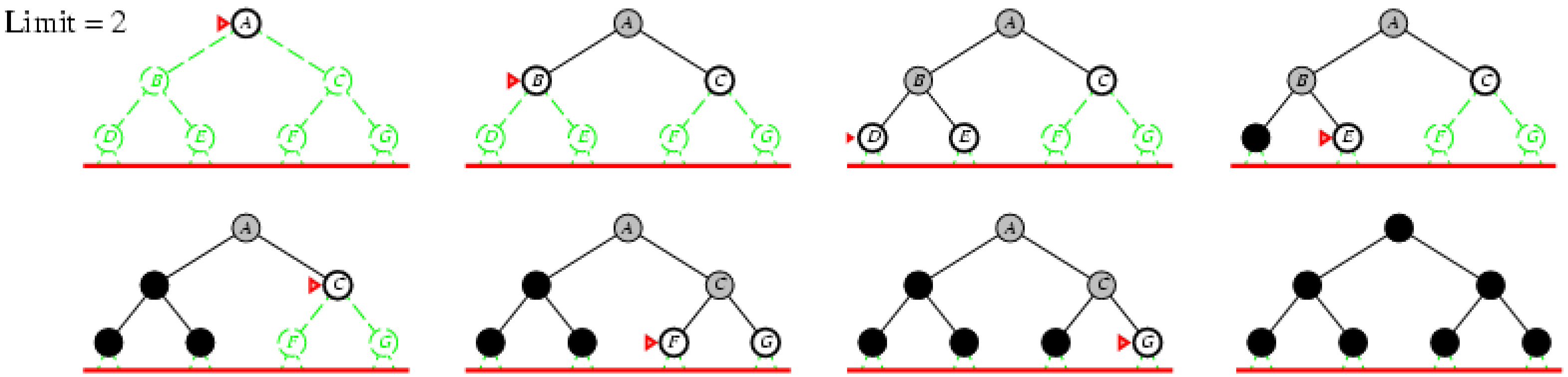
Iterative Deepening Search *k=0*

Limit = 0

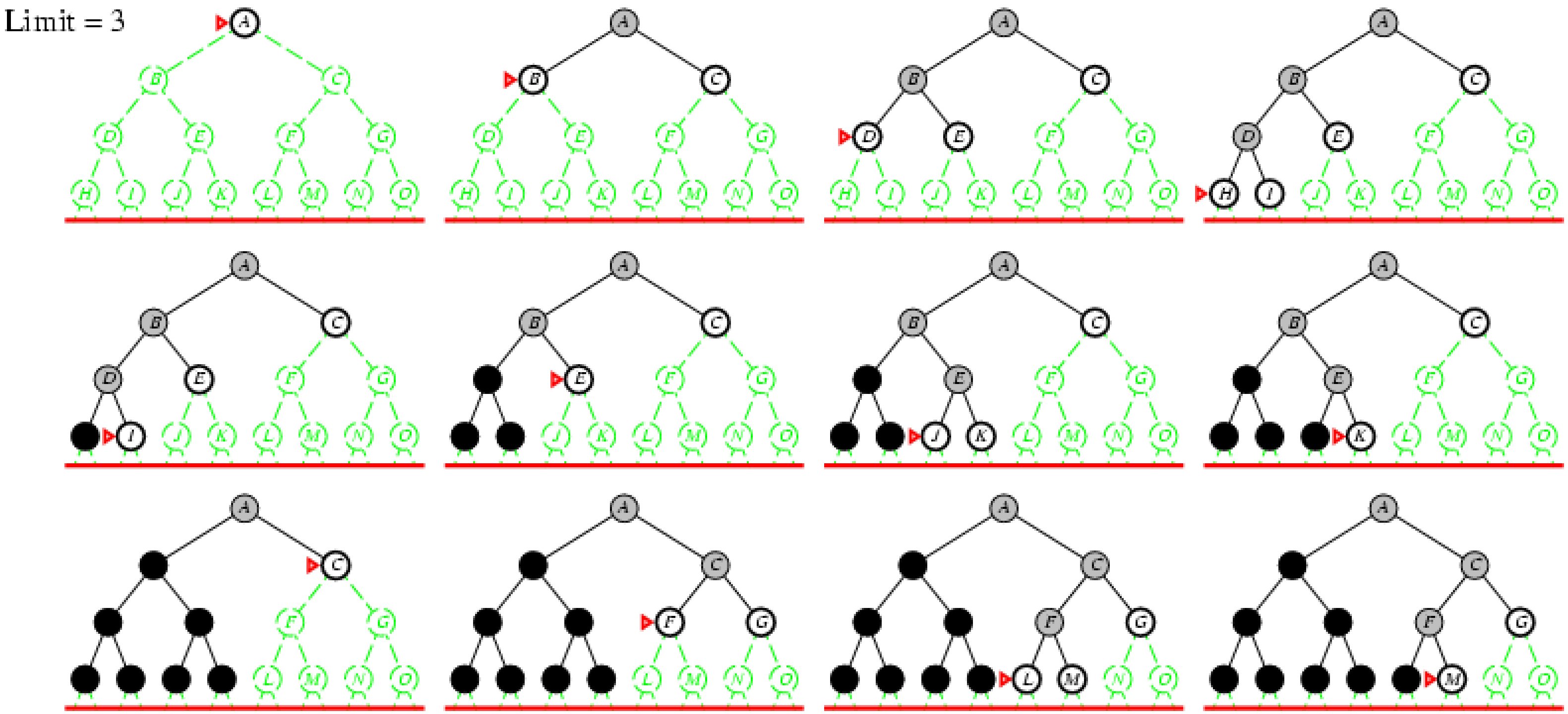
Iterative Deepening Search: *k=1*

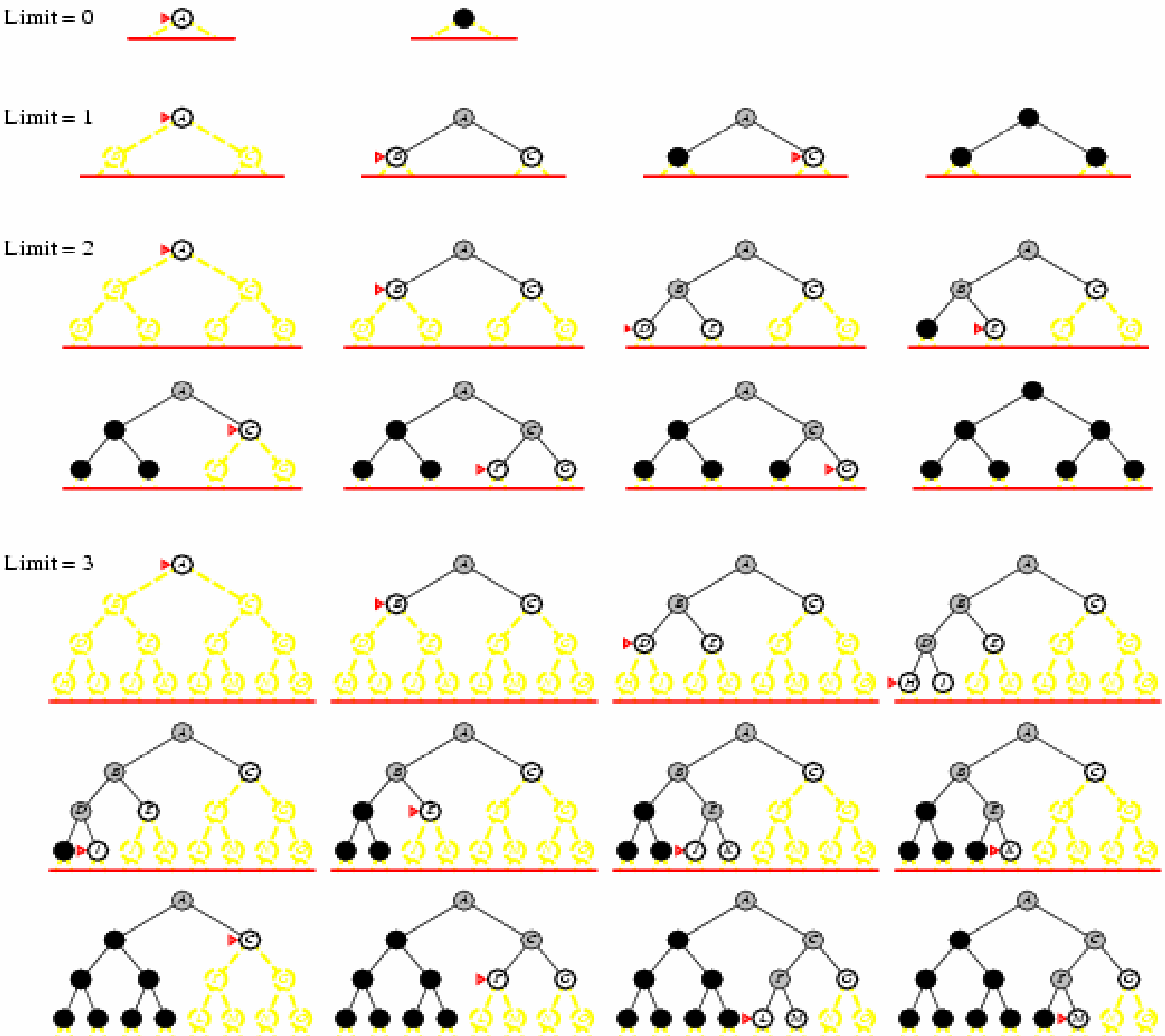


Iterative Deepening Search: *k* =2



Iterative Deepening Search *k=3*




Limit = 0
Limit = 1
Limit = 2
Limit = 3

Iterative Deepening: Analysis

- Time: $\approx$ BFS !
  - … even though it regenerates intermediate nodes !
- Why? Almost all work ON FINAL LEVEL anyway!
- Eg: b = 10, d = 5:
  - BFS expands 1 + 10 + 100 + … + 100,000 = 111,111
- IDS expands
  - bottom level: 1 time
  - second to bottom: 2 times
  - …
  - toplevel: d+1 times

total: $(d+1)b^0 + d\,b^1 + (d-1)b^2 + \ldots + 3b^{d-2} + 2b^{d-1} + 1b^d$

… 100,000 + 20,000 + …+ 50 + 6 = 123,456

- Ratio of IDS to BFS: $\approx [b / (b-1)]^2$
- Cost of repeating work at shallow depths: MINOR!

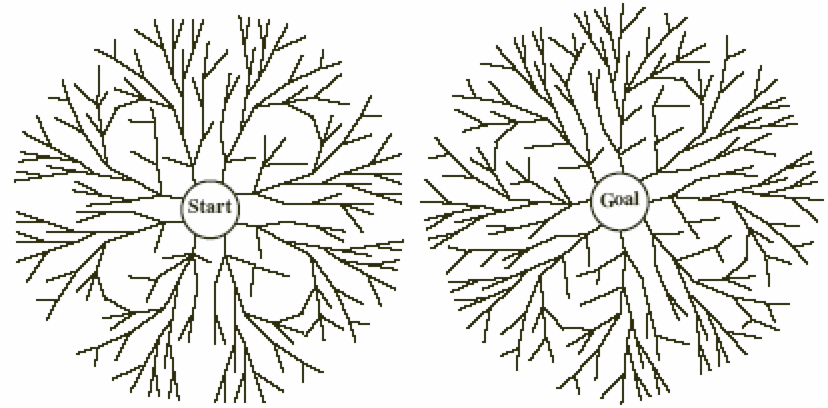
Properties of Iterative Deepening Search

- Complete? Yes
- Optimal? Yes, if step cost = 1
- Time?

  $(d+1)b^0 + d\, b^1 + (d-1)b^2 + \ldots + b^d = O(b^d)$
- Space? $O(b\, d)$

  - IDS does not get stuck on infinite path
  - Space: Same as DFS – but with *d*, not *m* (as each search is DFS)

BiDirectional Search

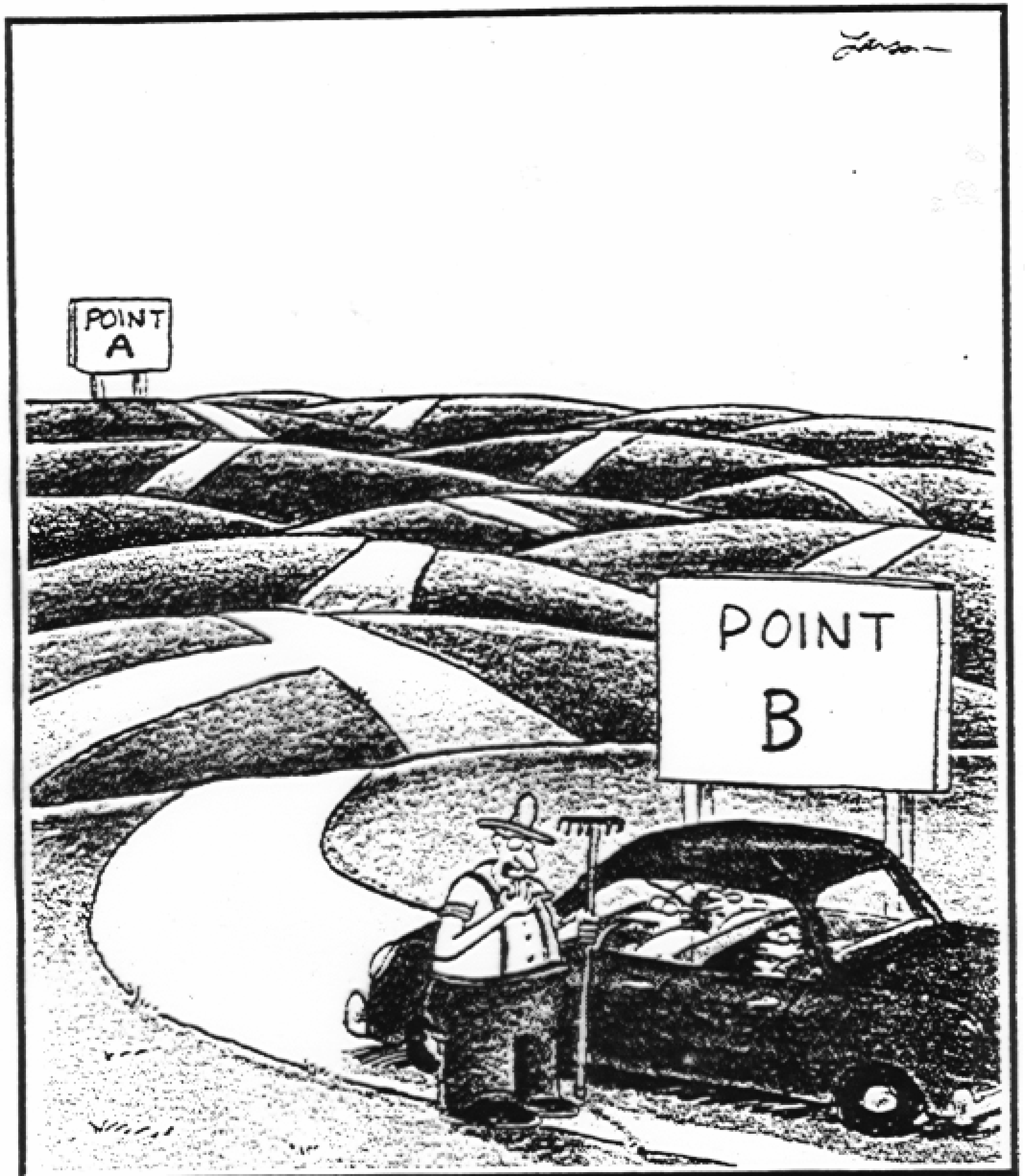


- Simultaneously:
  - Search "forward" from start
  - Search "backward" from goal

  Stop when two searches meet in middle
- If branching factor = b in each direction & solution at depth d
  $\Rightarrow$ need only $O(2b^{d/2}) = O(b^{d/2})$ steps

- Eg: b = 10, d = 6:

  BFS expands 1,111,111 nodes
  BiDirectional: 2,222 !
- Issues:
  - How to "search backwards from goal"?
  - What if > 1 goals (chess)?
  - How to check if paths meet? constant time?
  - What type of search done in each half? (BFS)



"Well, lemme think. ... You've stumped me, son.
Most folks only wanna know how
to go the other way."

Comparing “Blind" Search Strategies

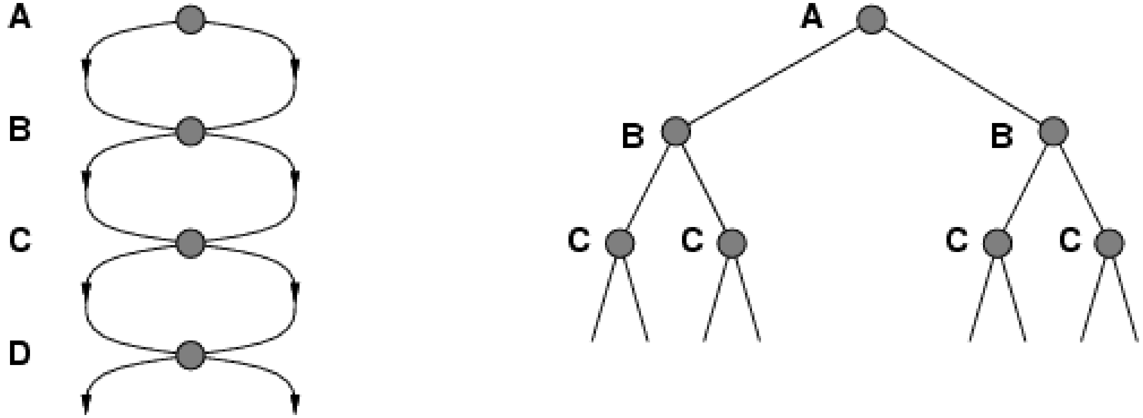
Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening
Complete?	Yes	Yes	No	No	Yes
Time	$O(b^{d+1})$	$O(b^{\lceil C^*/\epsilon \rceil})$	$O(b^m)$	$O(b^l)$	$O(b^d)$
Space	$O(b^{d+1})$	$O(b^{\lceil C^*/\epsilon \rceil})$	$O(bm)$	$O(bl)$	$O(bd)$
Optimal?	Yes	Yes	No	No	Yes

Comparison of Strategies

- Breadth-first is complete and optimal, but has high space complexity
  - Bad when branching factor is high
- Depth-first is space efficient, but not complete nor optimal
  - Bad when search depth is infinite
- Iterative deepening is asymptotically optimal !

Avoiding Repeated States

- May reach same state thru multiple paths



- ... if operations are REVERSIBLE (∞)
- ⇒ "Obvious" algorithms may
  - be inefficient (exponentially worse)
  - loop forever!

Repeated States

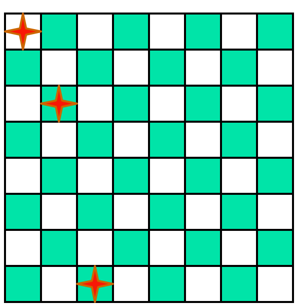
No

Few

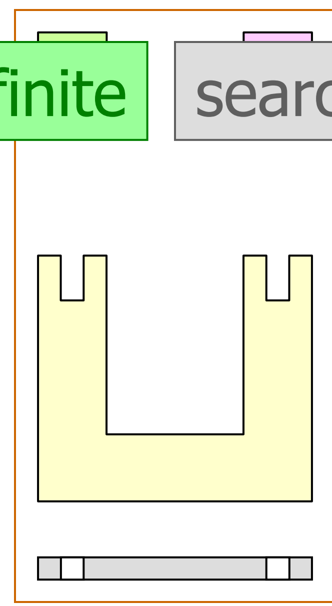
Many

search tree is finite

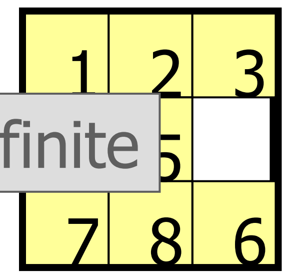
search tree is infinite

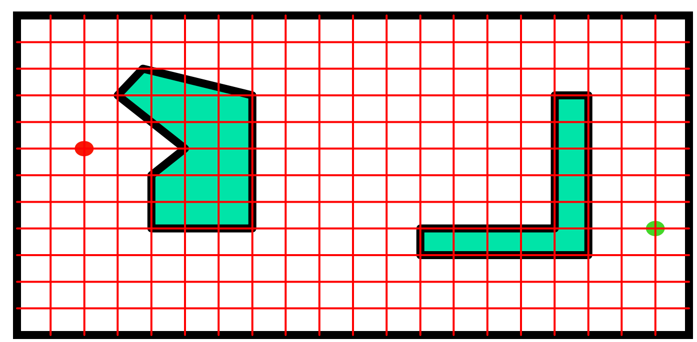


8-queens



assembly planning





8-puzzle and robot navigation

Approaches to Deal w/Repeated State

- Don't return to parent state
  - Don't generate successor ≡ node's parent
- Don't allow cycles
  - Don't generate successor ≡ node's ancestor
- Don't ever revisit state
  - Keep every visited state in memory! $O(b^d)$

Summary of Blind Search

- Search strategies:
  - breadth-first, depth-first, iterative deepening, …
- Evaluation of strategies:
  - completeness, optimality, time and space complexity
- Iterative deepening search
  - uses only linear space
  - ≈ same time as other blind-searchers
- Avoid repeated states