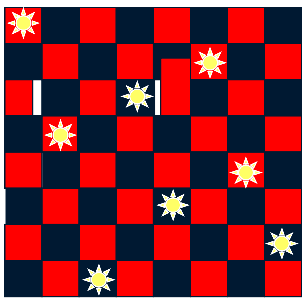
Search Space Problems

- State Space : Graph of states (Express constraints and parameters of the problem)
- Operators : Transformations applied to the states.
- Start state : S_0 (Search starts from here)
- Goal state(s) : $\{G\}$ - Search terminates here.
- Cost : Effort involved in using an operator.
- Optimal path : Least cost path

8-queen problem



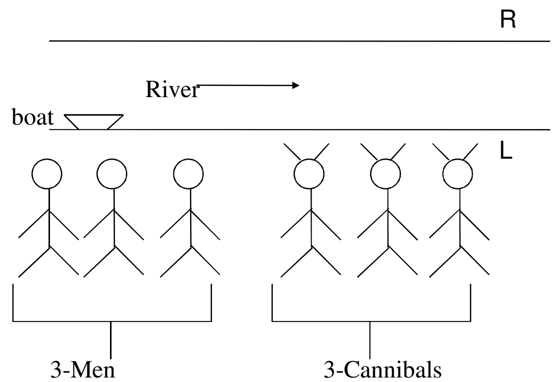
States: Any arrangements of 0 – 8 queens on board

Initial State: No queens on board

Successor function: Add a queen to any empty square

Goal State: 8 queens on the board, unattacked

Men and Cannibals



Constraints

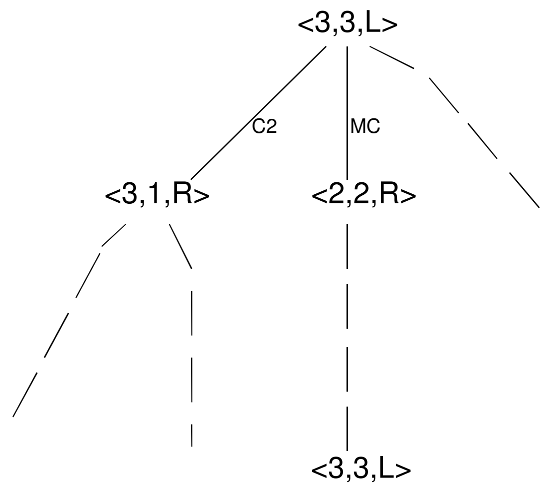
- The boat can carry at most 2 people
- On no bank should the cannibals outnumber the Men
- Move all people to the other side of the river

Men and Cannibals

State : *<#M, #C, P>*
#M = Number of men on bank *L*
#C = Number of cannibals on bank *L*
P = Position of the boat

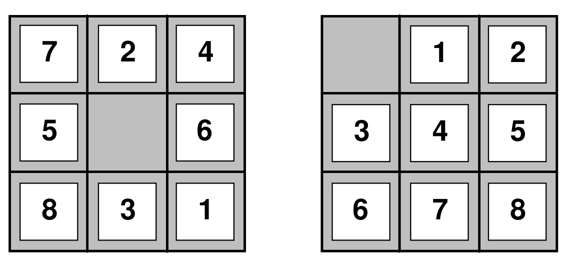
S0 = *<3, 3, L>*
G = *< 0, 0, R >*

Operations
M2 = Two men take boat
M1 = One man takes boat
C2 = Two cannibals take boat
C1 = One cannibal takes boat
MC = One man and one cannibal takes boat



Search tree

8 -puzzle



Start State **Goal State**

Standard for the problem... Tile movement represented as the movement of the blank space.

Operators:

L : Blank moves left

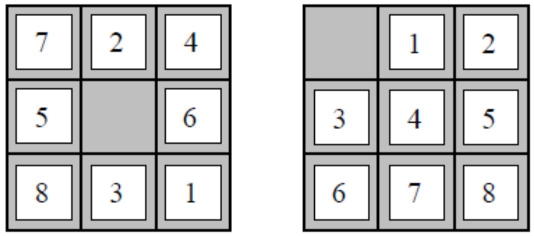
R : Blank moves right

U : Blank moves up

D : Blank moves down

$C(L) = C(R) = C(U) = C(D) = 1$

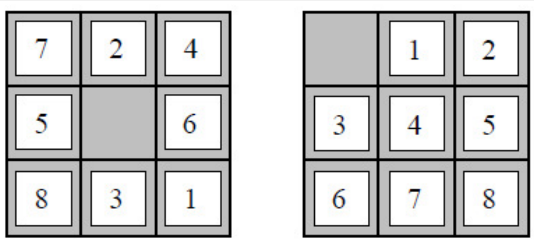
Example: The 8-puzzle



Start State   Goal State

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

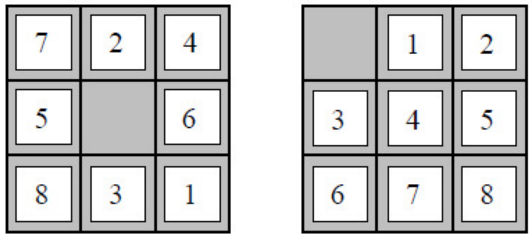
Example: The 8-puzzle



Start State   Goal State

states??: integer locations of tiles (ignore intermediate positions)
actions??
goal test??
path cost??

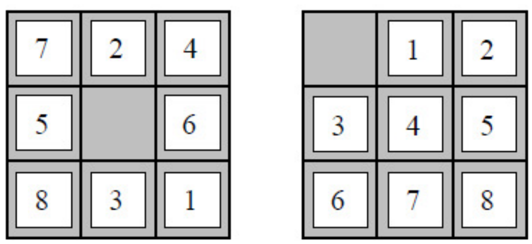
Example: The 8-puzzle



Start State Goal State

states??: integer locations of tiles (ignore intermediate positions)
actions??: move blank left, right, up, down (ignore unjamming etc.)
goal test??
path cost??

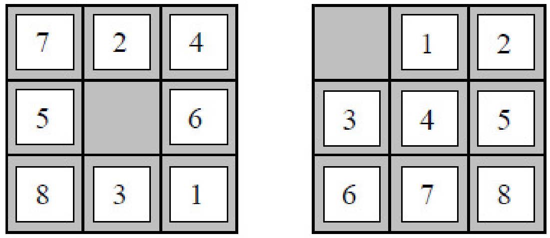
Example: The 8-puzzle



Start State Goal State

states??: integer locations of tiles (ignore intermediate positions)
actions??: move blank left, right, up, down (ignore unjamming etc.)
goal test??: = goal state (given)
path cost??

Example: The 8-puzzle

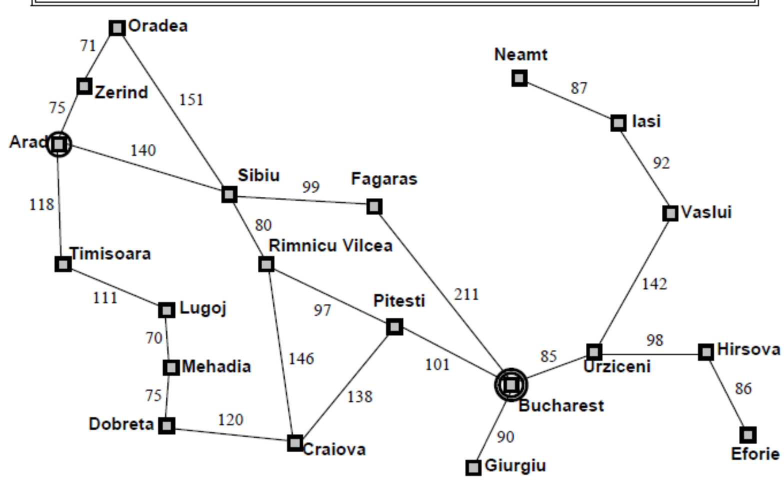


Start State | Goal State

states??: integer locations of tiles (ignore intermediate positions)
actions??: move blank left, right, up, down (ignore unjamming etc.)
goal test??: = goal state (given)
path cost??: 1 per move

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

Example: Romania



Example: Romania

On holiday in Romania; currently in Arad.
Flight leaves tomorrow from Bucharest

Formulate goal:
  be in Bucharest

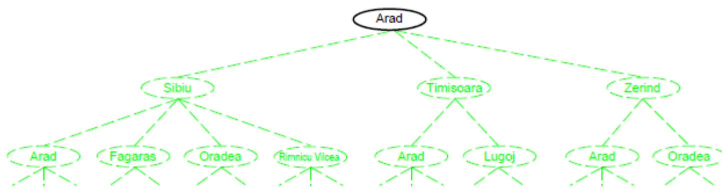
Formulate problem:
  states: various cities
  actions: drive between cities

Find solution:
  sequence of cities, e.g., Arad, Sibiu, Fagaras, Bucharest

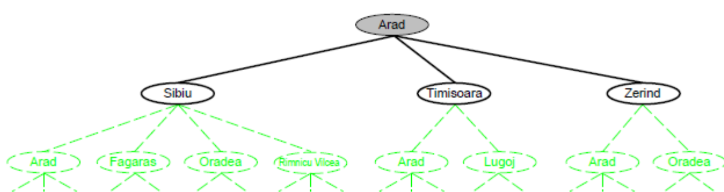
Example: Romania

- Initial State: at Arad
- States Space: being at any city
- Successor function: set of state-pairs
  S(Arad) = Zerind
- Goal State: Bucharest
- Path cost : sum of distances
- Solution : sequence of states

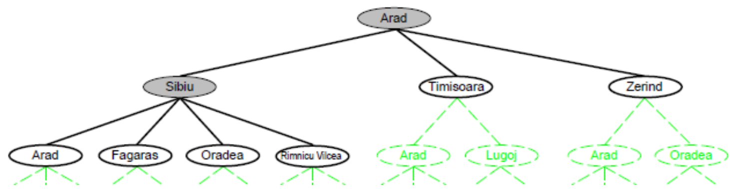
Example: Romania



Example: Romania



Example: Romania

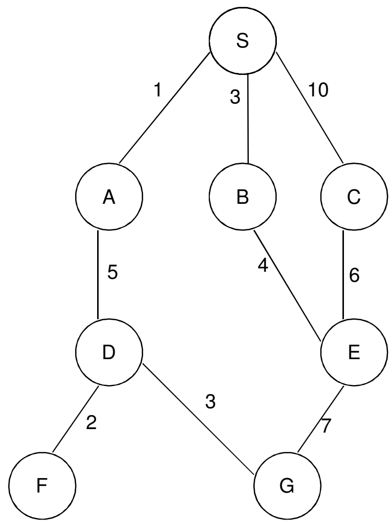


Tree Search Algorithm

- Basic Idea: Simulated exploration of state space by generating successors of already explored states

```
function TREE-SEARCH( problem, strategy) returns a solution, or failure
   initialize the search tree using the initial state of problem
   loop do
      if there are no candidates for expansion then return failure
      choose a leaf node for expansion according to strategy
      if the node contains a goal state then return the corresponding solution
      else expand the node and add the resulting nodes to the search tree
   end
```

General Graph search Algorithm



Graph G = (V,E)

1) Open List : S $^{(\emptyset, 0)}$
Closed list : Ø

2) OL : $A^{(S,1)}$, $B^{(S,3)}$, $C^{(S,10)}$
CL : S

3) OL : $B^{(S,3)}$, $C^{(S,10)}$, $D^{(A,6)}$
CL : S, A

4) OL : $C^{(S,10)}$, $D^{(A,6)}$, $E^{(B,7)}$
CL: S, A, B

5) OL : $D^{(A,6)}$, $E^{(B,7)}$
CL : S, A, B , C

6) OL : $E^{(B,7)}$, $F^{(D,8)}$, $G^{(D, 9)}$
CL : S, A, B, C, D

7) OL : $F^{(D,8)}$, $G^{(D,9)}$
CL : S, A, B, C, D, E

8) OL : $G^{(D,9)}$
CL : S, A, B, C, D, E, F

9) OL : Ø
CL : S, A, B, C, D, E,
F, G

Steps of GGS

1. Create a search graph *G*, consisting only of the start node *S*; put *S* on a list called *OPEN.*

2. Create a list called *CLOSED* that is initially empty.

3. Loop: if *OPEN* is empty, exit with failure.

4. Select the first node on *OPEN*, remove from *OPEN* and put on *CLOSED*, call this node *n*.

5. if *n* is the goal node, exit with the solution obtained by tracing a path along the pointers from *n* to *s* in *G*.

6. Expand node *n*, generating the set *M* of its successors that are not ancestors of *n*.

GGS steps (contd.)

7. Establish a pointer to *n* from those members of *M* that were not already in *G* (*i.e.*, not already on either *OPEN* or *CLOSED*). Add these members of *M* to *OPEN*. For each member of *M* that was already on *OPEN* or *CLOSED*, decide whether or not to redirect its pointer to *n*. For each member of M already on *CLOSED*, decide for each of its descendents in *G* whether or not to redirect its pointer.

8. Reorder the list *OPEN* using some strategy.

9. Go *LOOP.*

Search Strategies

Uninformed/Blind Search

- Breadth First Search
- Depth First Search
- Depth Limited Search
- Bidirectional Search

Informed/Heuristic Search

- Hill Climbing Search
- A* Algorithm

Measuring problem-Solving performance

A strategy is defined by picking the order of node expansion

What makes one search scheme better than another?

Completeness: Guarantee to find a solution?

Time complexity: How long is it to find a sol. (# of nodes)?

Optimality: Does the strategy find the shortest path (note some books use least cost)?

Space complexity: How much memory is needed (max. # of nodes in memory)?

Notations

- b: Branching Factor that is maximum number of successors of any node
- d : depth of the least cost solution
- C* : path cost of the optimal solution
- m : maximum depth of the state space

Breadth First Search

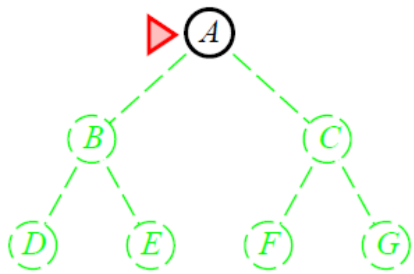
- Simple Strategy
- The root is expanded first, Then all its successors, Then all their successors
- At a given depth, All nodes are expanded.
- With branching factor b, at level d, we have

$1+b+b^2+b^3+...b^d + b(b^d -1) = O(b^{d+1})$ Nodes

- At level 12 with branching factor 10, we have 10^{13} nodes
- Space Problem !

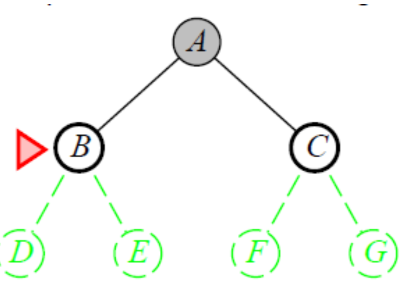
Breadth First Search

- Expand the shallowest node



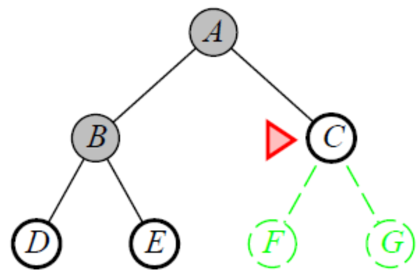
Breadth First Search

- Expand the shallowest node



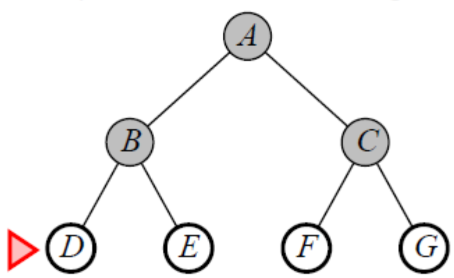
Breadth First Search

- Expand the shallowest node



Breadth First Search

- Expand the shallowest node



BFS

Completeness?

Yes, if solution exists, there is a guarantee to find it

Time complexity?

$O(b^{d+1})$

Space complexity?

$O(b^{d+1})$ *:* keeps every node in memory

Optimality?

Yes :finds shortest path

Remark:

If the definition of optimality is to find lowest cost path them BFS is not optimal

Bidirectional Search

BFS in both directions

How could this help?

b^{d+1} vs $2b^{(d+1)/2}$

- Can reduce time complexity,
- Not always applicable
- May require lots of space
- Hard to implement

Bidirectional Search

Completeness?

Yes, if solution exists, there is a guarantee to find it

Time complexity?

$O(b^{(d+1)/2})$, b is branching factor, d is least cost to goal

Space complexity?

$O(b^{(d+1)/2})$

Optimality?

yes

Uniform Cost Search

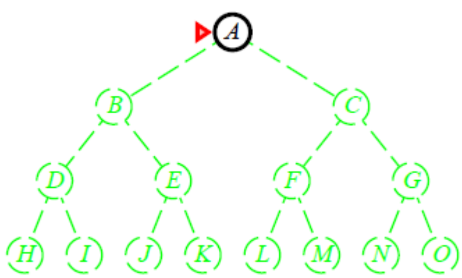
- Instead of expanding the shallowest node(like BFS), **uniform-cost search** expands the node n with the *lowest path cost*
- *Will be optimal according to the lowest cost definition*
- Uniform-cost search is guided by path costs rather than depths, so its complexity cannot easily be characterized in terms of *b and d. If is measured in terms of the optimal path C**

Depth First Search

- Always expand deepest node in the fringe of the tree.
- Modest memory requirement, stores only single path from root to leaf.
- With branching factor b, at level d, we store only bm+1 i.e. O(bm)
- It may stuck in an infinite path and never finds solution

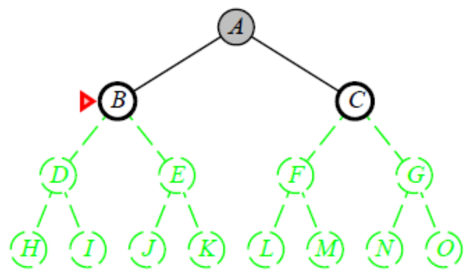
Depth First Search

- Expand deepest unexpanded node



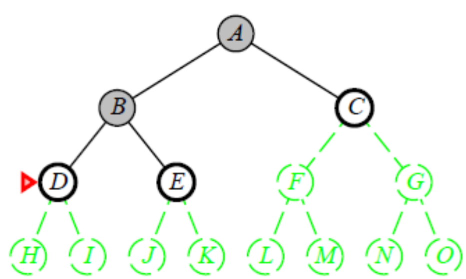
Depth First Search

- Expand deepest unexpanded node



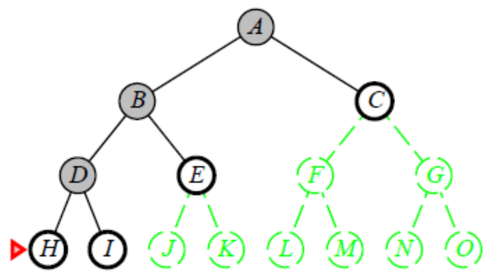
Depth First Search

- Expand deepest unexpanded node



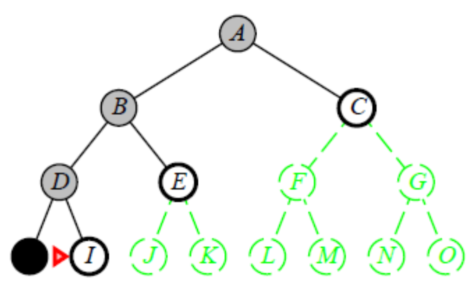
Depth First Search

- Expand deepest unexpanded node



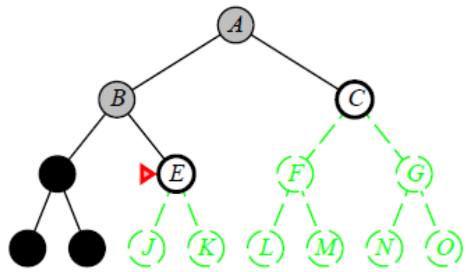
Depth First Search

- Expand deepest unexpanded node



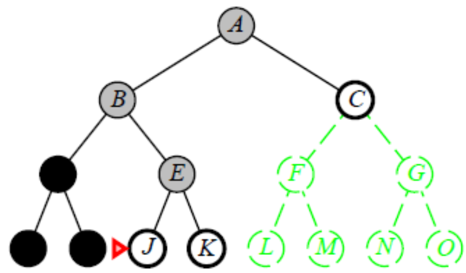
Depth First Search

- Expand deepest unexpanded node



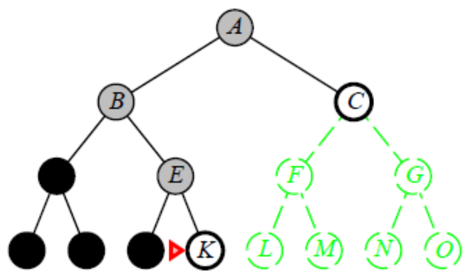
Depth First Search

- Expand deepest unexpanded node



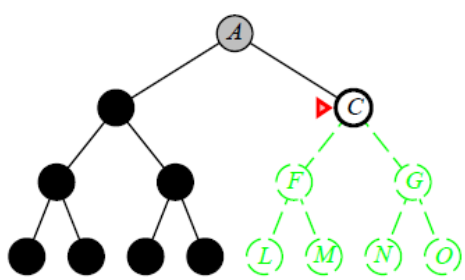
Depth First Search

- Expand deepest unexpanded node



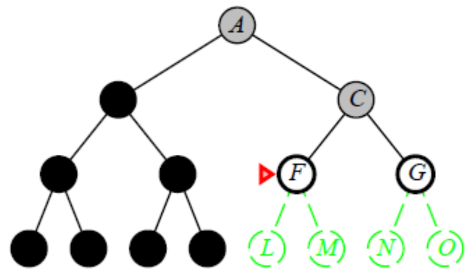
Depth First Search

- Expand deepest unexpanded node



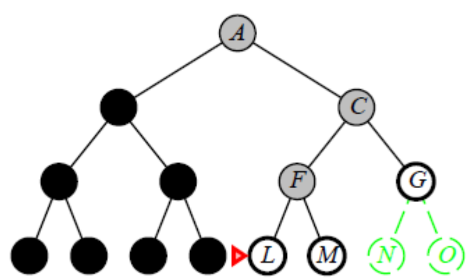
Depth First Search

- Expand deepest unexpanded node



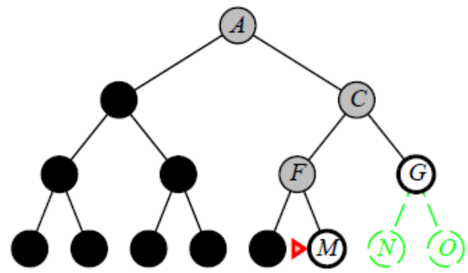
Depth First Search

- Expand deepest unexpanded node



Depth First Search

- Expand deepest unexpanded node



DFS

Completeness?

No, fails in infinite depth spaces or spaces with loops

Yes, assuming state space finite.

Time complexity?

O(b^m), terrible if *m* is much bigger than d. can do well if lots of goals

Space complexity?

O(bm), i.e. linear

Optimality?

No may find a solution with long path

Depth-limited Search

Put a limit to the level of the tree

DFS, only expand nodes depth ≤ L.

Completeness?

No, if L ≤ d.

Time complexity?

$O(b^L)$

Space complexity?

$O(bL)$

Optimality?

No

Iterative Deepening

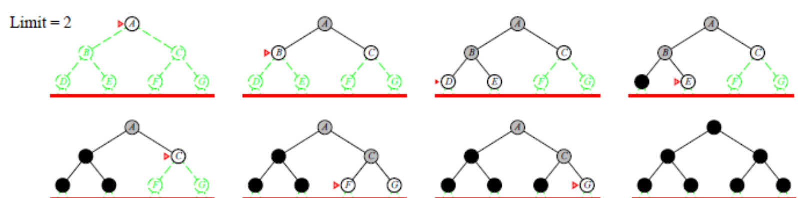
- Calls depth-limited search with increasing limits until goal is found

Iterative Deepening

- Calls depth-limited search with increasing limits until goal is found

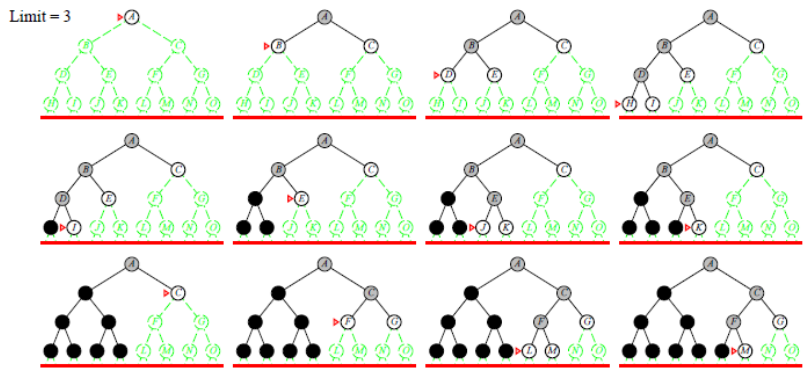
Iterative Deepening

- Calls depth-limited search with increasing limits until goal is found



Iterative Deepening

- Calls depth-limited search with increasing limits until goal is found



Iterative Deepening

- Completeness?

  Yes.

  Time complexity?

  $O(b^d) = (d+1)\, b^0 + d b^1 + \ldots + b^d$

  Space complexity?

  $O(bd)$

  Optimality?

  Yes; if looking for shortest path

  Remark: IDS performs much faster than BFS:

  Numerical comparison for $b = 10$ and $d = 5$, solution at far right leaf:

$$N(\text{IDS}) = 50 + 400 + 3,000 + 20,000 + 100,000 = 123,450$$
$$N(\text{BFS}) = 10 + 100 + 1,000 + 10,000 + 100,000 + 999,990 = 1,111,100$$

Remarks

- BFS works as a queue. Pick the leftmost element of the open list , evaluate it and add its children to the end of the list, FIFO
- DFS works as a stack. Pick the leftmost element of the open list , evaluate it and add its children to the beginning of the list, LIFO

Informed Search

- Blind search - no notion concept of the "right direction"
  - can only recognize goal once it's achieved
- Heuristic search – we have rough idea of how good various states are, and use this knowledge to guide our search
- Can find solutions more efficient than uninformed
- General approach is *best-first-search*
- A node is selected based on an *evaluation function f(n)*
- A node that **seems** to be best is picked and it may not be the actual best