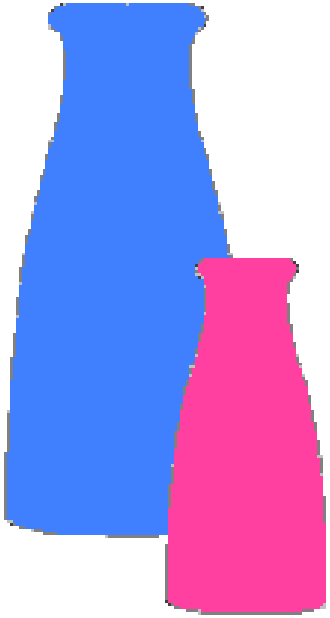
State-Space Searches

State spaces

- A state space consists of
  - A (possibly infinite) set of states
    - The start state represents the initial problem
    - Each state represents some configuration reachable from the start state
    - Some states may be goal states (solutions)
  - A set of operators
    - Applying an operator to a state transforms it to another state in the state space
    - Not all operators are applicable to all states
- State spaces are used extensively in Artificial Intelligence (AI)

State Space: Water Jug Problem

"You are given two jugs, a 4-litre one and a 3-litre one. Neither has any measuring markers on it. There is a pump that can be used to fill the jugs with water. How can you get exactly 2 litres of water into 4-litre jug."



State Space: Water Jug Problem

- State: (x, y)

  $x = 0, 1, 2, 3, \text{ or } 4$ $\qquad\qquad$ $y = 0, 1, 2, 3$

- Start state: $(0, 0)$.
- Goal state: $(2, n)$ for any n.
- Attempting to end up in a goal state.

State Space: Water Jug Problem

1. $(x, y) \rightarrow (4, y)$
   if $x < 4$

2. $(x, y) \rightarrow (x, 3)$
   if $y < 3$

3. $(x, y) \rightarrow (x - d, y)$
   if $x > 0$

4. $(x, y) \rightarrow (x, y - d)$
   if $y > 0$

State Space: Water Jug Problem

5. $(x, y) \rightarrow (0, y)$
   if $x > 0$

6. $(x, y) \rightarrow (x, 0)$
   if $y > 0$

7. $(x, y) \rightarrow (4, y - (4 - x))$
   if $x + y \geq 4, y > 0$

8. $(x, y) \rightarrow (x - (3 - y), 3)$
   if $x + y \geq 3, x > 0$

State Space: Water Jug Problem

9. $(x, y) \rightarrow (x + y, 0)$

   if $x + y \leq 4, y > 0$

10. $(x, y) \rightarrow (0, x + y)$

    if $x + y \leq 3, x > 0$

11. $(0, 2) \rightarrow (2, 0)$

12. $(2, y) \rightarrow (0, y)$

State Space Search: Water Jug Problem

1. current state = (0, 0)

2. Loop until reaching the goal state (2, 0)
   - Apply a rule whose left side matches the current state
   - Set the new current state to be the resulting state

(0, 0)
(0, 3)
(3, 0)
(3, 3)
(4, 2)
(0, 2)
(2, 0)

Example 2: The 15-puzzle

Start state:

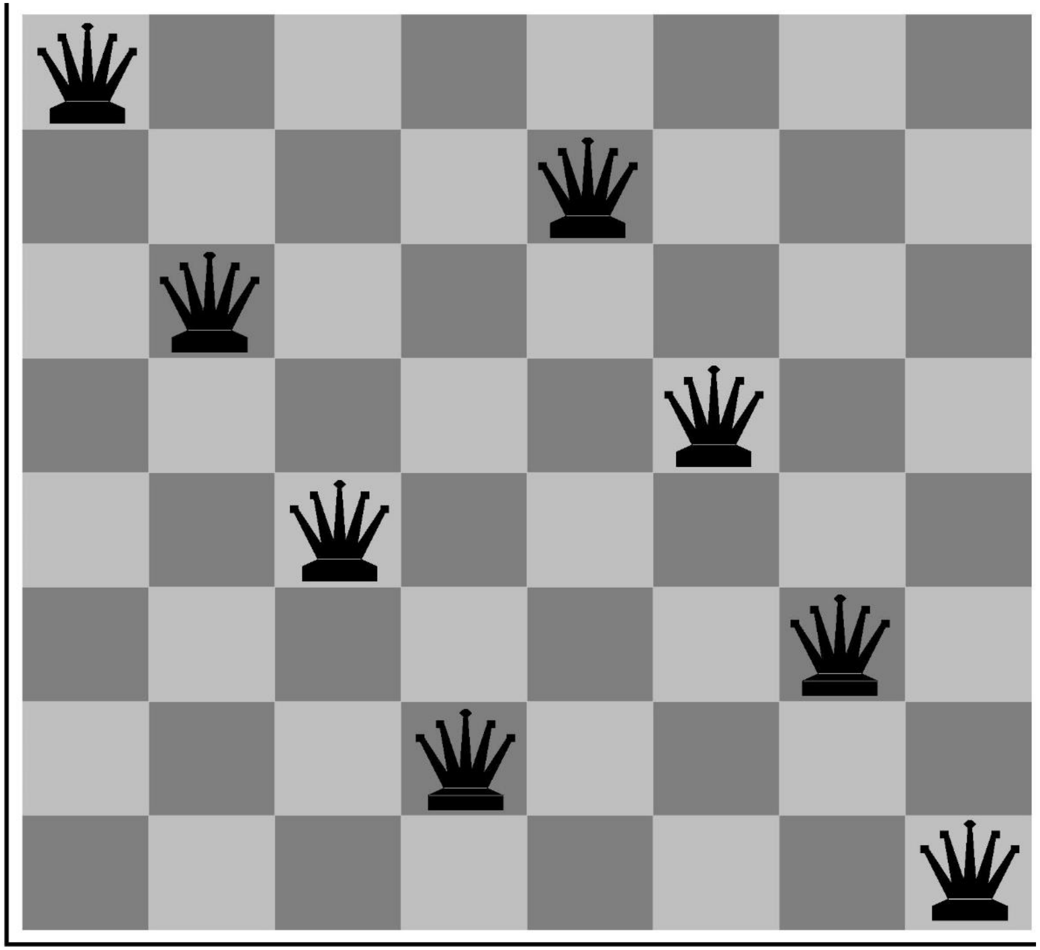
3	10	13	7
9	14	6	1
4		15	2
11	8	5	12

Goal state:

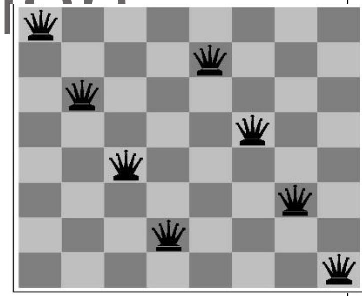
1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

- The start state is some (almost) random configuration of the tiles
- The goal state is as shown
- Operators are
  - Move empty space up
  - Move empty space down
  - Move empty space right
  - Move empty space left
- Operators apply if not against edge

8 Queen Problem



State-Space problem formulation



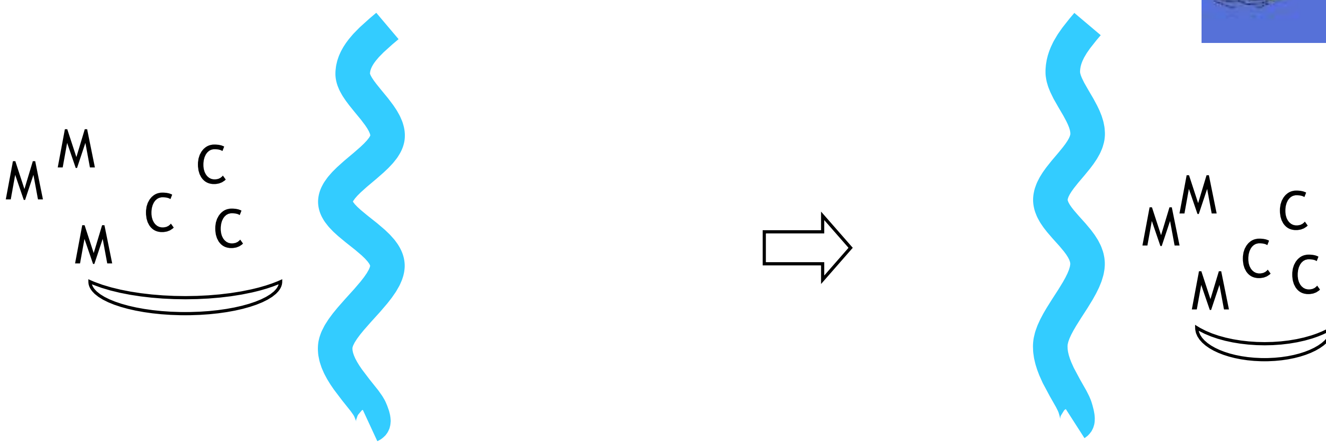
- states? -any arrangement of n<=8 queens

  -*or* arrangements of n<=8 queens in leftmost n columns, 1 per column, such that no queen attacks any other.
- initial state? no queens on the board
- actions? -add queen to any empty square

  -*or* add queen to leftmost empty square such that it is not attacked by other queens.
- goal test? 8 queens on the board, none attacked.
- path cost? 1 per move

Example 3: Missionaries and cannibals

- An old puzzle is the "Missionaries and cannibals" problem (in various guises)
- The missionaries and cannibals wish to cross a river
- They have a canoe that can hold two people
- It is unsafe to have cannibals outnumber missionaries



Initial state

Goal state

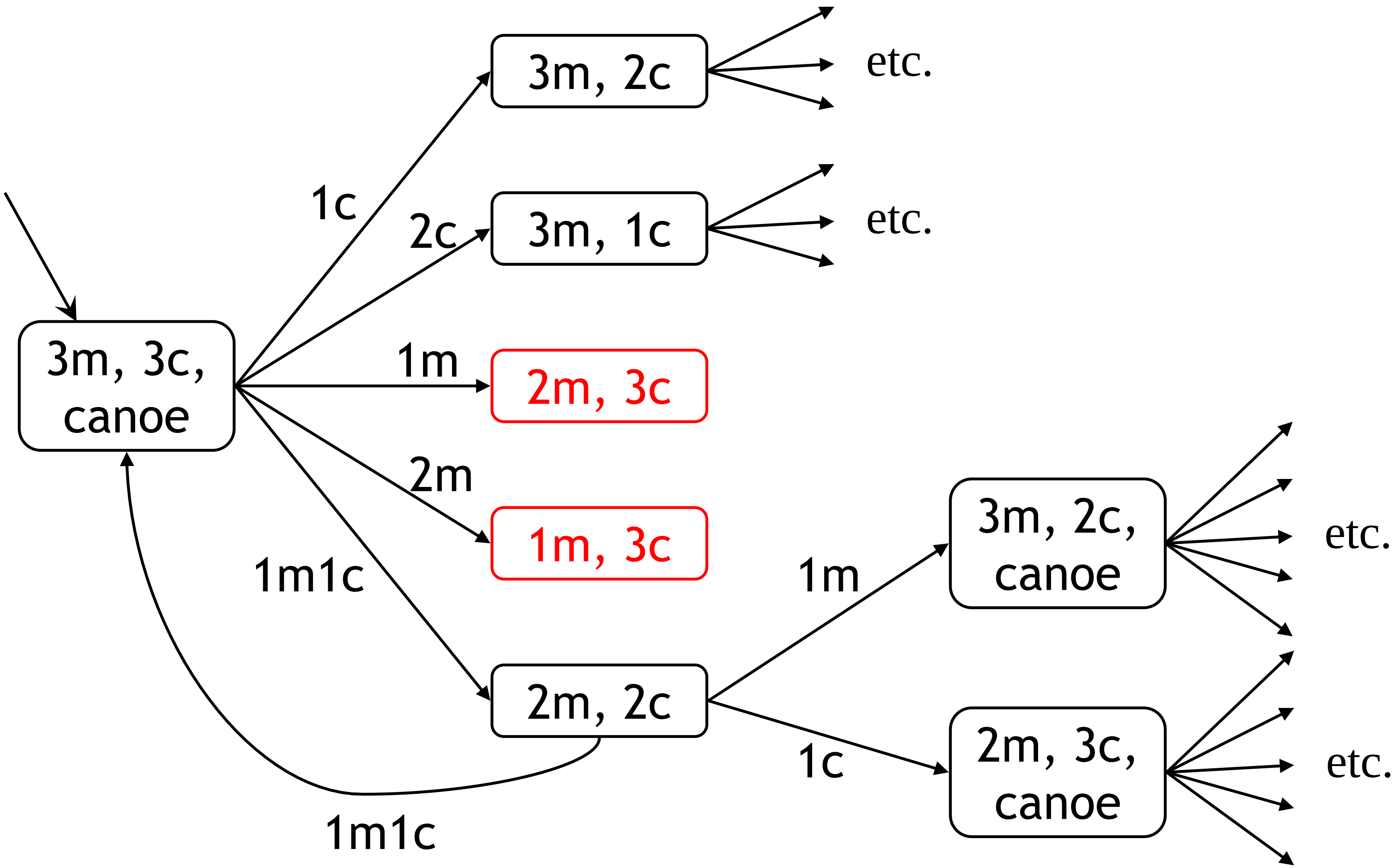
States

- A *state* can be represented by the number of missionaries and cannibals on each side of the river
  - Initial state: `3m,3c,canoe / 0m,0c`
  - Goal state: `0m,0c / 3m,3c,canoe`
  - We assume that crossing the river is a simple procedure that always works (so we don't have to represent the canoe being in the middle of the river)
- However, this is redundant; we only need to represent how many missionaries/cannibals are on *one* side of the river
  - Initial state: `3m,3c,canoe`
  - Goal state: `0m,0c`

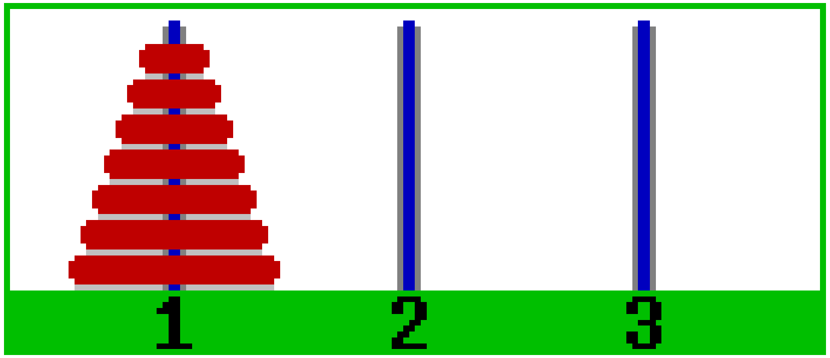
Operations

- An *operation* takes us from one state to another
- Here are five possible operations:
  - Canoe takes 1 missionary across river (1m)
  - Canoe takes 1 cannibal across river (1c)
  - Canoe takes 2 missionaries across river (2m)
  - Canoe takes 2 cannibals across river (2c)
  - Canoe takes 1 missionary and 1 cannibal across river (1m1c)
- We don't have to specify "west to east" or "east to west" because only one of these will be possible at any given time

The state space



Example Problems – Towers of Hanoi



States: combinations of poles and disks

Operators: move disk x from pole y to pole z subject to constraints

• cannot move disk on top of smaller disk
• cannot move disk if other disks on top

Goal test: disks from largest (at bottom) to smallest on goal pole

Path cost: 1 per move

Towers of Hanoi applet

Example Problems – Rubik's Cube

States: list of colors for each cell on each face

Initial state: one specific cube configuration

Operators: rotate row x or column y on face z direction a

Goal: configuration has only one color on each face

Path cost: 1 per move

Rubik's cube applet

State-space searching

- Most problems in AI can be cast as searches on a state space
- The space can be tree-shaped or graph-shaped
  - If a graph, need some way to keep track of where you have been, so as to avoid loops
- The state space is often very, very large
- We can minimize the size of the search space by careful choice of operators
- Exhaustive searches don't work—we need *heuristics*

Sample Search Problems

- Graph coloring
- Protein folding
- Game playing
- Airline travel
- Proving algebraic equalities
- Robot motion planning

The basic search algorithm

Initialize: put the start node into OPEN
while OPEN is not empty
    take a node N from OPEN
    if N is a goal node, report success
    put the children of N onto OPEN
Report failure

- If OPEN is a stack, this is a depth-first search
- If OPEN is a queue, this is a breadth-first search
- If OPEN is a *priority queue*, sorted according to *most promising first*, we have a best-first search

Uninformed Search

A General State-Space Search Algorithm

- Node n
  - state description
  - parent (may use a backpointer) (if needed)
  - Operator used to generate n (optional)
  - Depth of n (optional)
  - Path cost from S to n (if available)
- OPEN list
  - initialization: {S}
  - node insertion/removal depends on specific search strategy
- CLOSED list
  - initialization: {}
  - organized by backpointers

A General State-Space Search Algorithm

```
open := {S}; closed :={};
repeat
   n := select(open);    /* select one node from open for expansion */
    if n is a goal
        then exit with success;  /* delayed goal testing */
    expand(n)
            /* generate all children of n
               put these newly generated nodes in open (check duplicates)
               put n in closed (check duplicates) */
until open = {};
exit with failure
```

Some Issues

- Search process constructs a search tree, where
  - **root** is the initial state S, and
  - **leaf nodes** are nodes
    - not yet been expanded (i.e., they are in OPEN list) or
    - having no successors (i.e., they're "deadends")
- Search tree may be infinite because of loops even if state space is small
- Search strategies mainly differ on *select* (open)
- Each node represents a partial solution path (and cost of the partial solution path) from the start node to the given node.
  - in general, from this node there are many possible paths (and therefore solutions) that have this partial path as a prefix.

Evaluating Search Strategies

- **Completeness**
  - Guarantees finding a solution whenever one exists
- **Time Complexity**
  - How long (worst or average case) does it take to find a solution? Usually measured in terms of the **number of nodes expanded**
- **Space Complexity**
  - How much space is used by the algorithm? Usually measured in terms of the **maximum size that the "OPEN" list** becomes during the search
- **Optimality/Admissibility**
  - If a solution is found, is it guaranteed to be an optimal one? For example, is it the one with minimum cost?

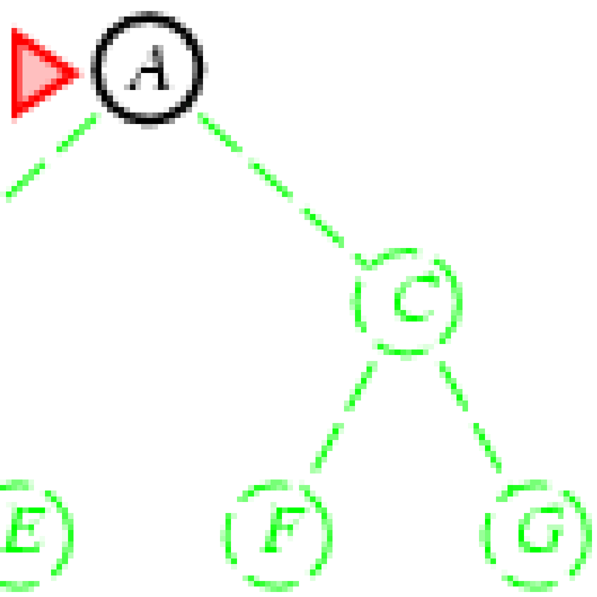
Uninformed search strategies

- Uninformed: While searching you have no clue whether one non-goal state is better than any other. Your search is blind.
- Various blind strategies:
- Breadth-first search
- Uniform-cost search
- Depth-first search
- Iterative deepening search

Breadth-first search

- Expand shallowest unexpanded node
- Implementation:
  - *fringe* is a first-in-first-out (FIFO) queue, i.e., new successors go at end of the queue.

Is A a goal state?

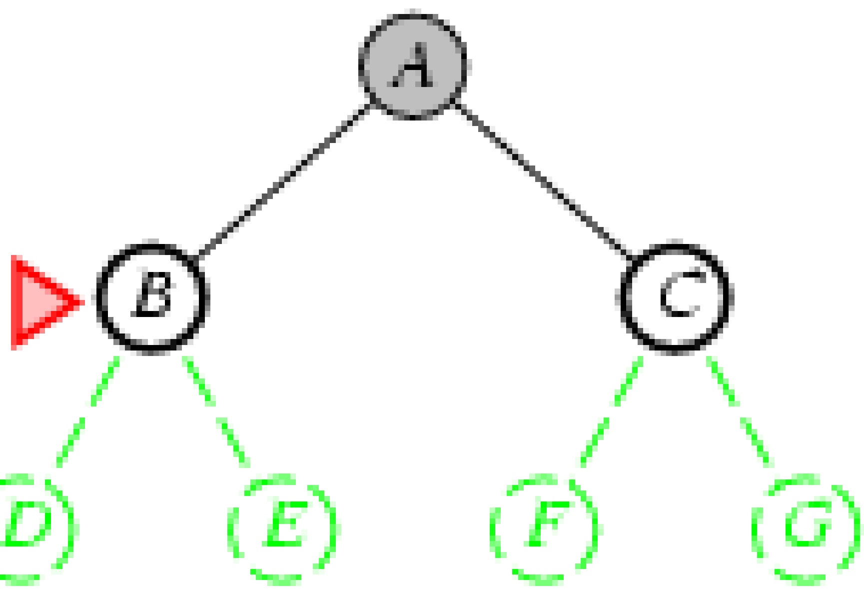


Breadth-first search

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

Expand:
fringe = [B,C]

Is B a goal state?

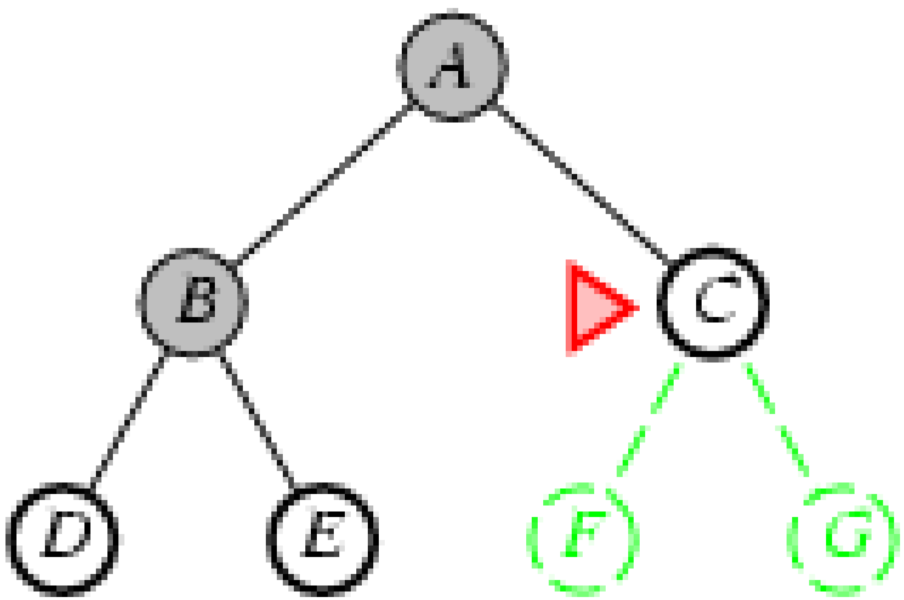


Breadth-first search

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

Expand:
fringe=[C,D,E]

Is C a goal state?

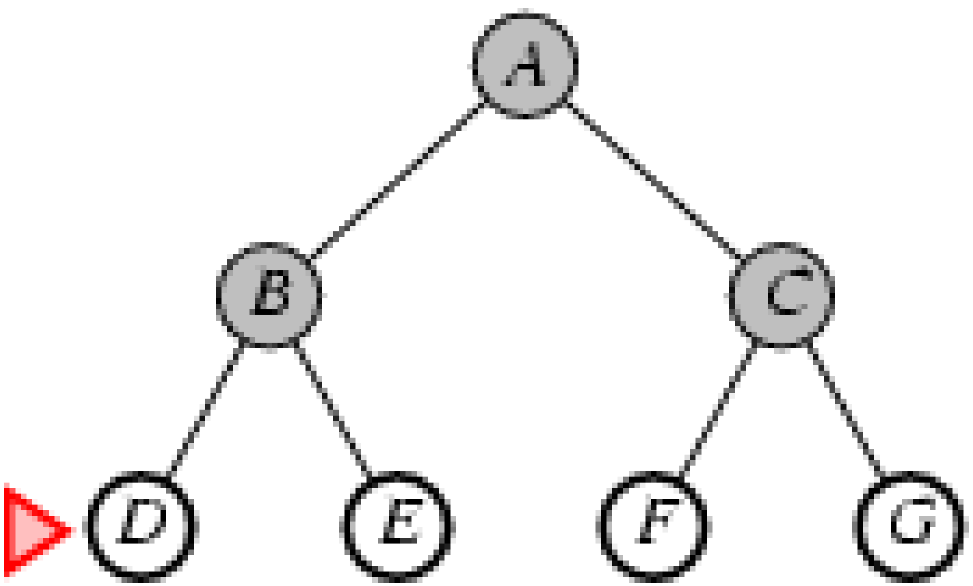


Breadth-first search

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

Expand:
fringe=[D,E,F,G]

Is D a goal state?



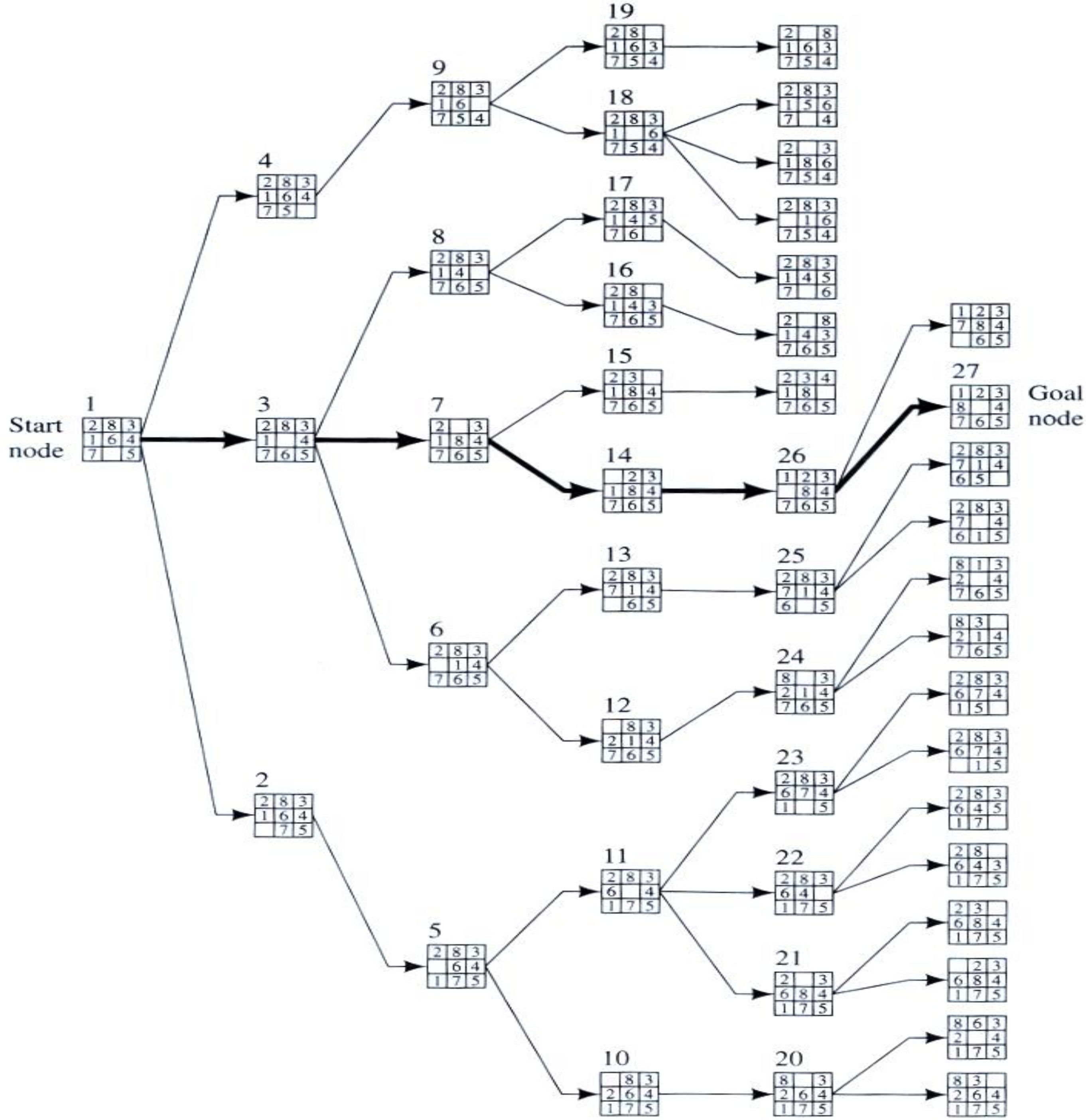
8 Puzzel Game

2	8	3
1	6	4
7		5

2		8
1	6	3
7	5	4

Initial State	**Goal State**

Example BFS



Start node
Goal node
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27

Analysis

- See what happens with b=10
  - expand 10,000 nodes/second
  - 1,000 bytes/node

Depth	Nodes	Time	Memory
2	1110	.11 seconds	1 megabyte
4	111,100	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
15	10^{15}	3,523 years	1 exabyte

Properties of breadth-first search

- Complete? Yes it always reaches goal (if b is finite)
- Time? $1+b+b^2+b^3+\ldots+b^d + (b^{d+1}-b)) = O(b^{d+1})$

  (this is the number of nodes we generate)
- Space? $O(b^{d+1})$ (keeps every node in memory,

  either in fringe or on a path to fringe).
- Optimal? Yes (if we guarantee that deeper solutions are less optimal, e.g. step-cost=1).

- Space is the bigger problem (more than time)

Uniform-cost search

Breadth-first is only optimal if step costs is increasing with depth (e.g. constant). Can we guarantee optimality for any step cost?

Uniform-cost Search: Expand node with smallest path cost g(n).

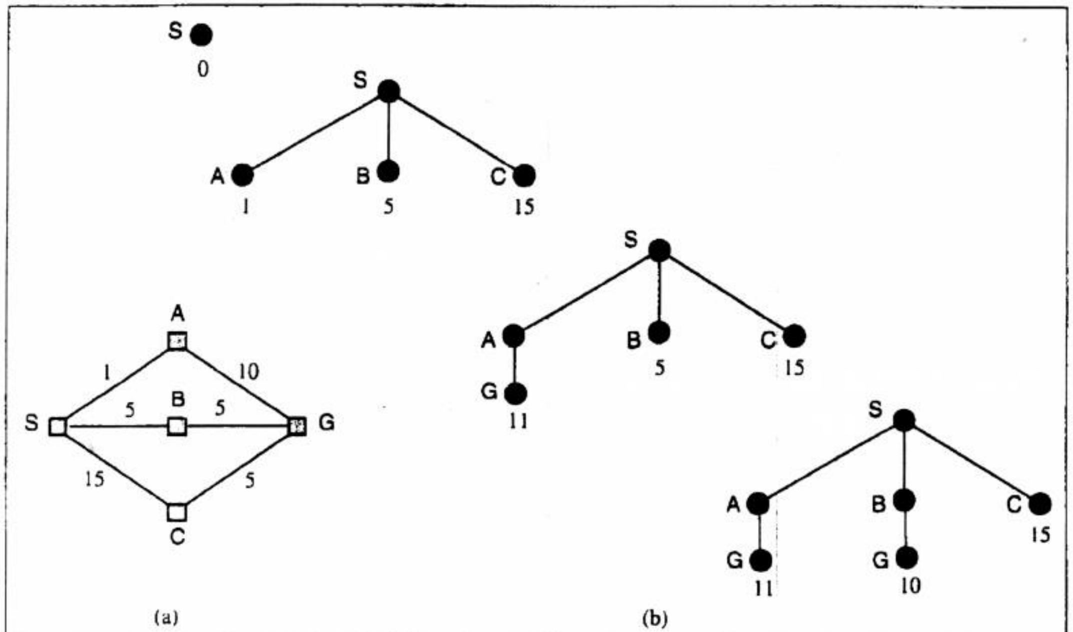
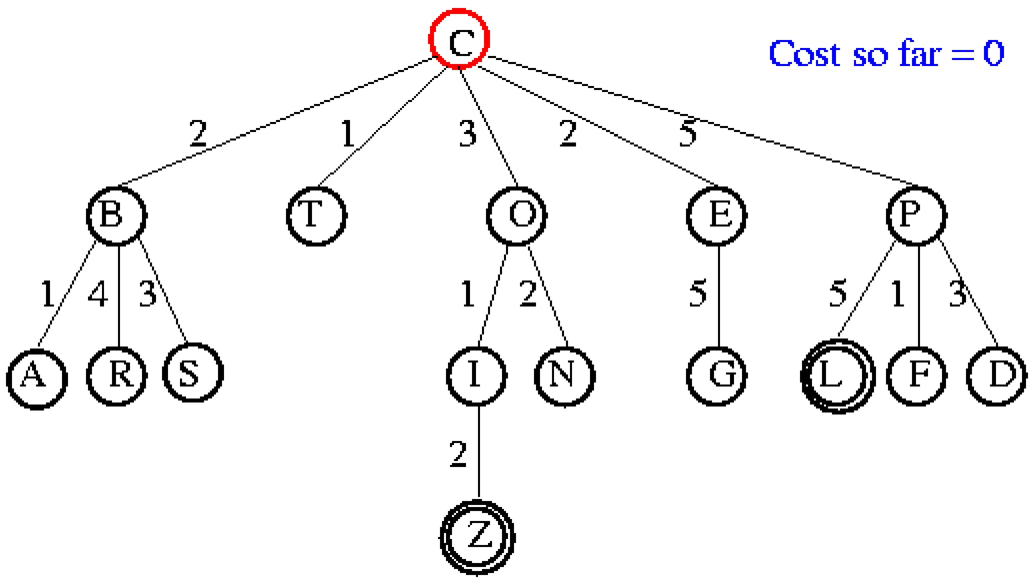


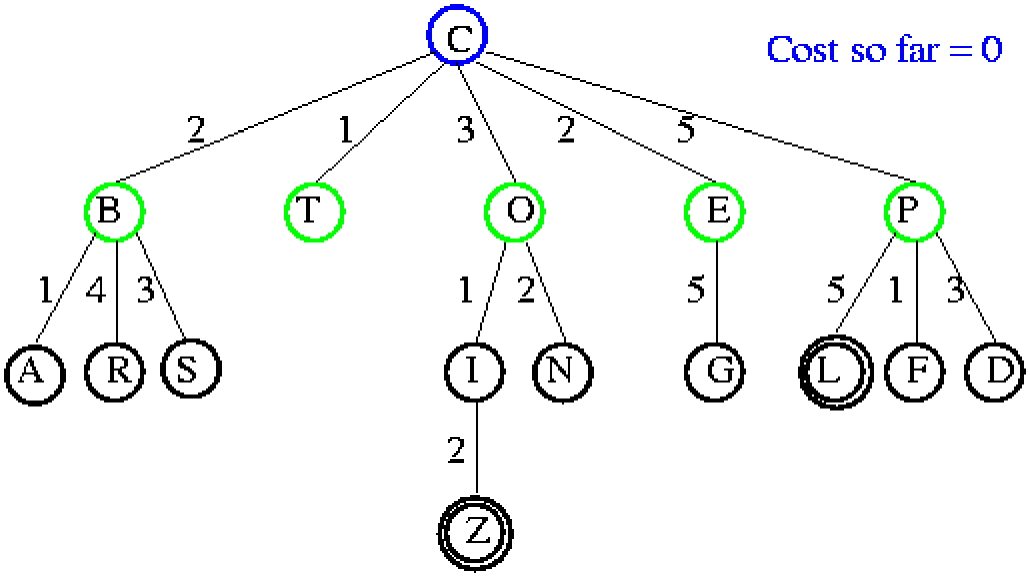
Figure 3.13 A route-finding problem. (a) The state space, showing the cost for each operator. (b) Progression of the search. Each node is labelled with $g(n)$. At the next step, the goal node with $g = 10$ will be selected.

UCS Example



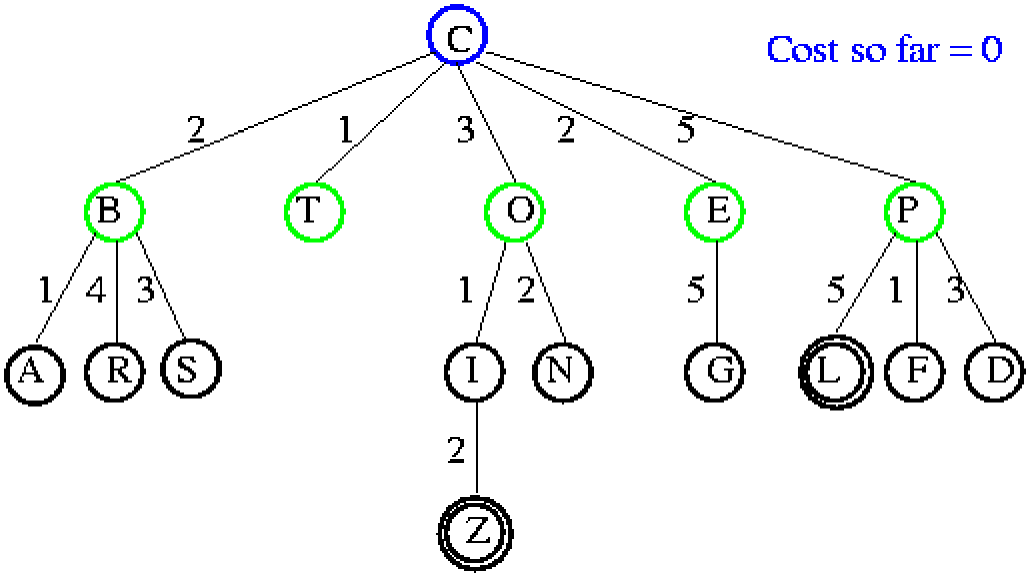
Open list: C

UCS Example



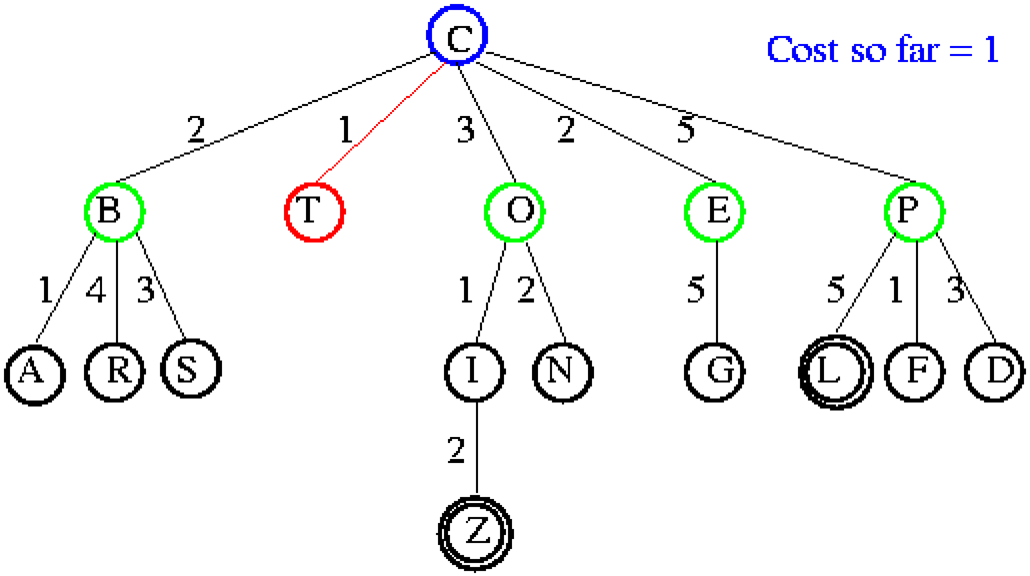
Open list: B(2) T(1) O(3) E(2) P(5)

UCS Example



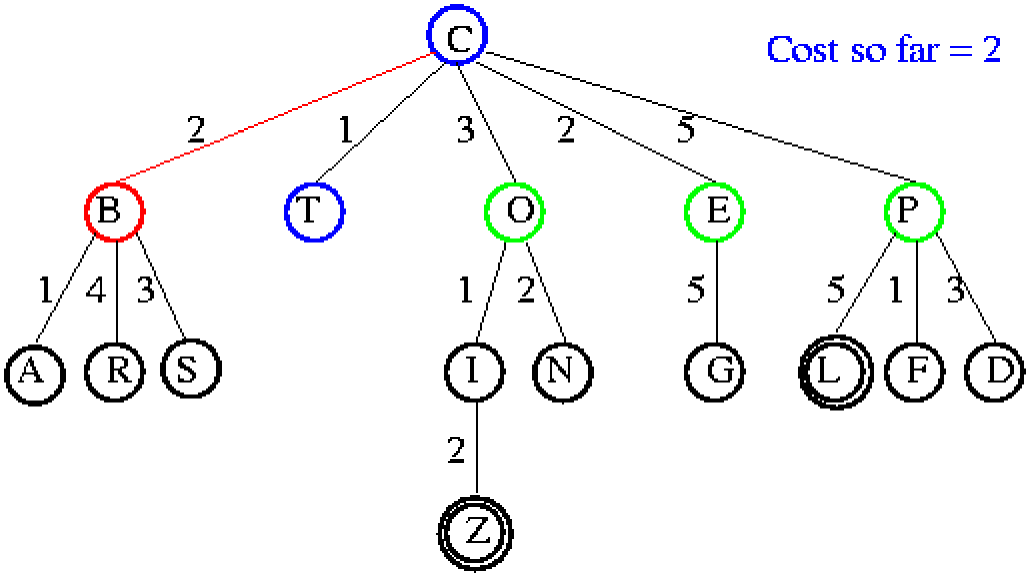
Open list: T(1) B(2) E(2) O(3) P(5)

UCS Example



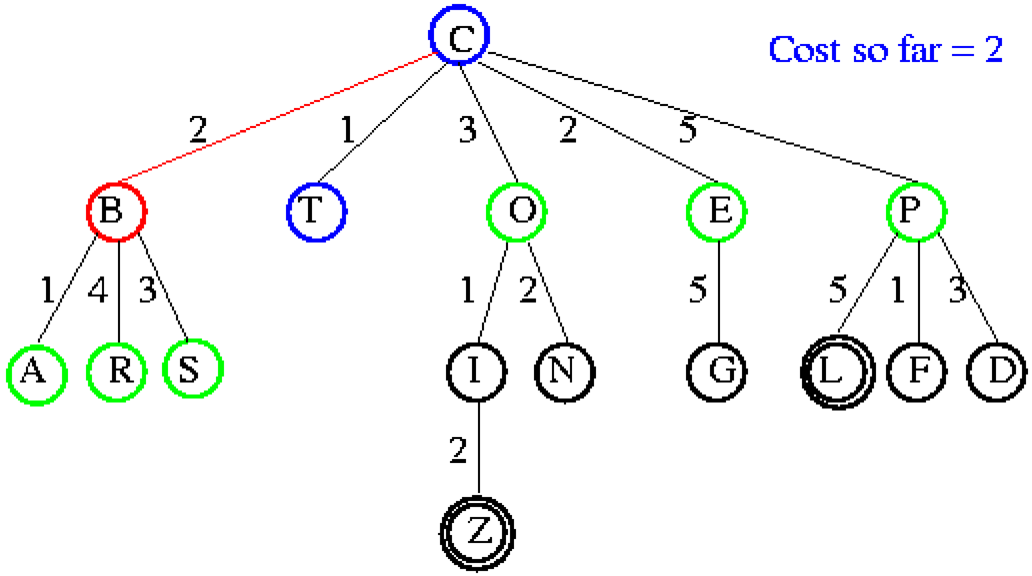
Open list: B(2) E(2) O(3) P(5)

UCS Example



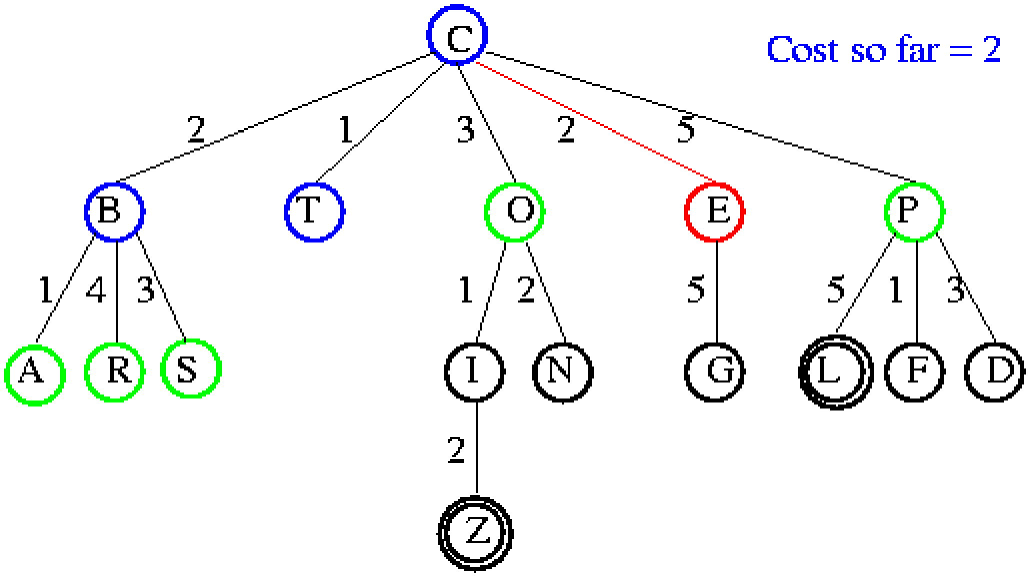
Open list: E(2) O(3) P(5)

UCS Example



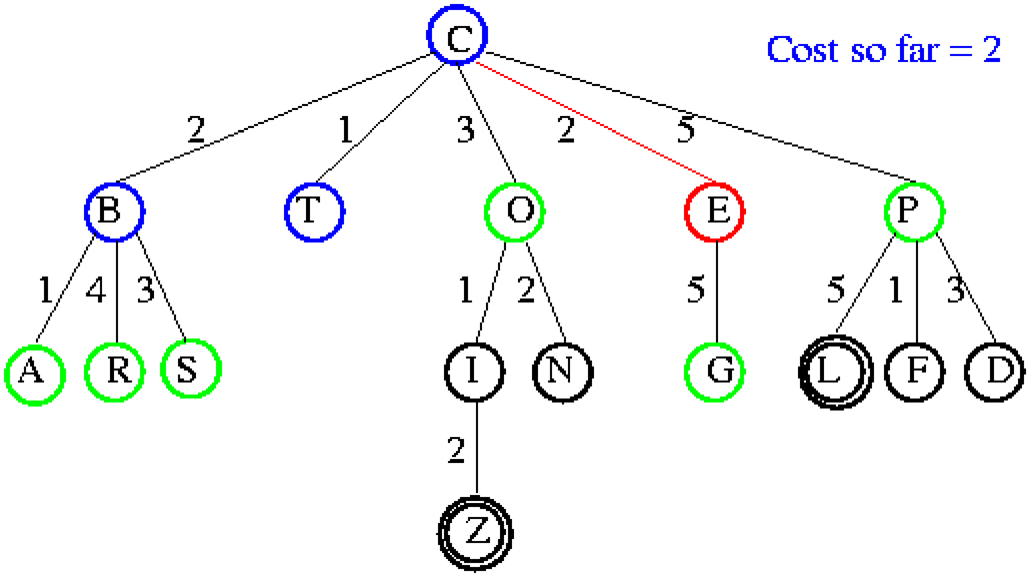
Open list: E(2) O(3) A(3) S(5) P(5) R(6)

UCS Example



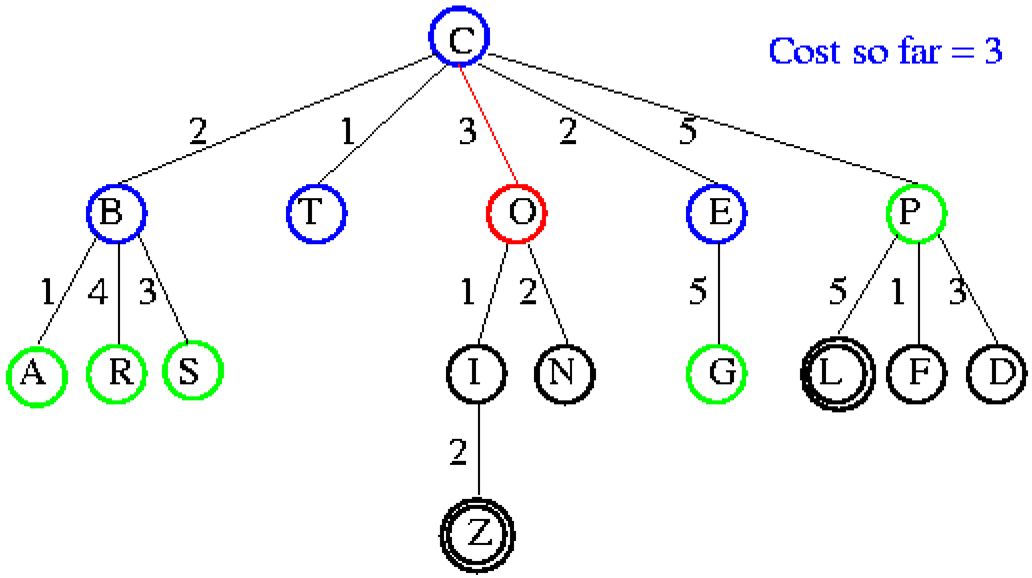
Open list: O(3) A(3) S(5) P(5) R(6)

UCS Example



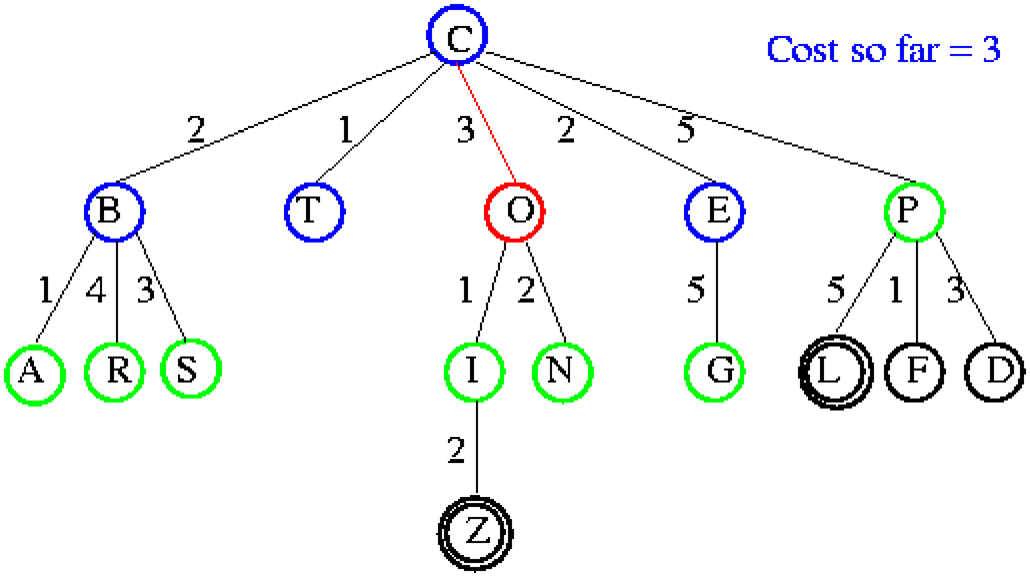
Open list: O(3) A(3) S(5) P(5) R(6) G(10)

UCS Example



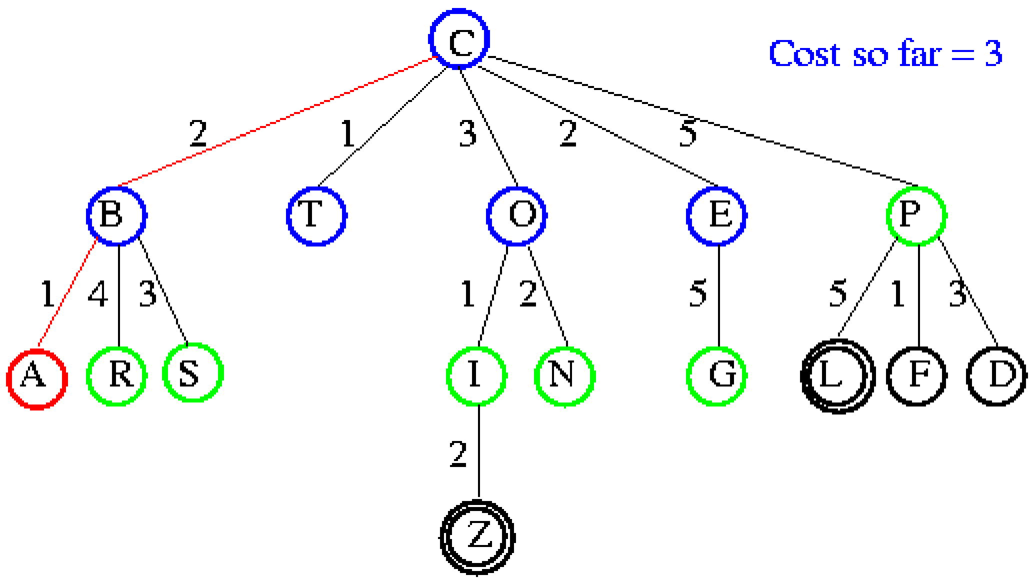
Open list: A(3) S(5) P(5) R(6) G(10)

UCS Example



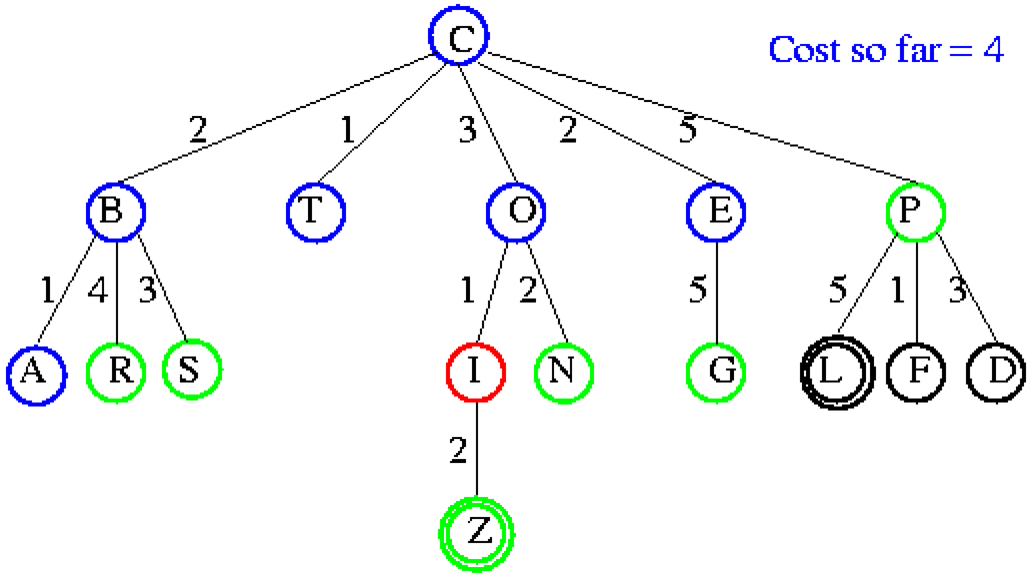
Open list: A(3) I(4) S(5) N(5) P(5) R(6) G(10)

UCS Example



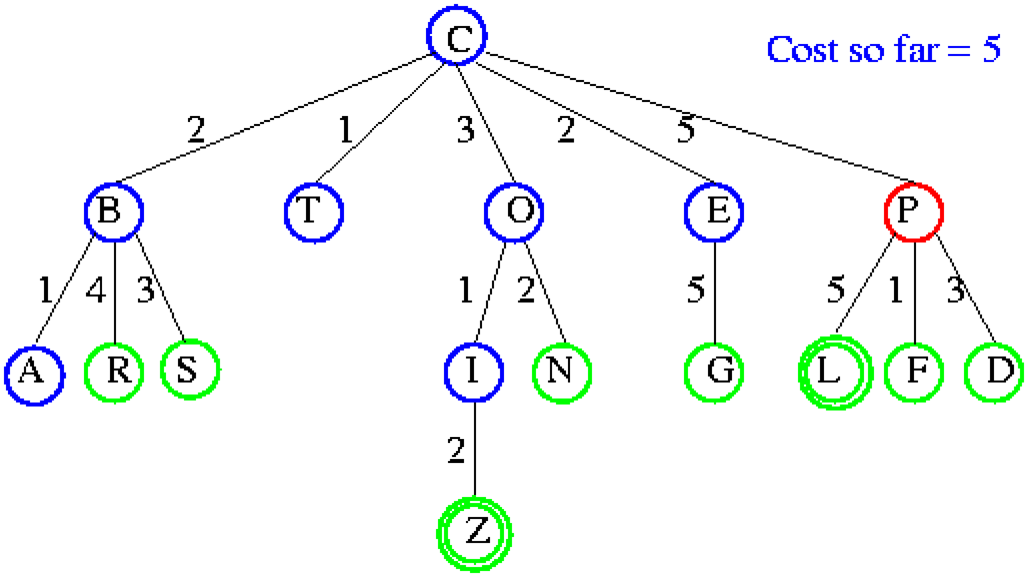
Open list: I(4) P(5) S(5) N(5) R(6) G(10)

UCS Example



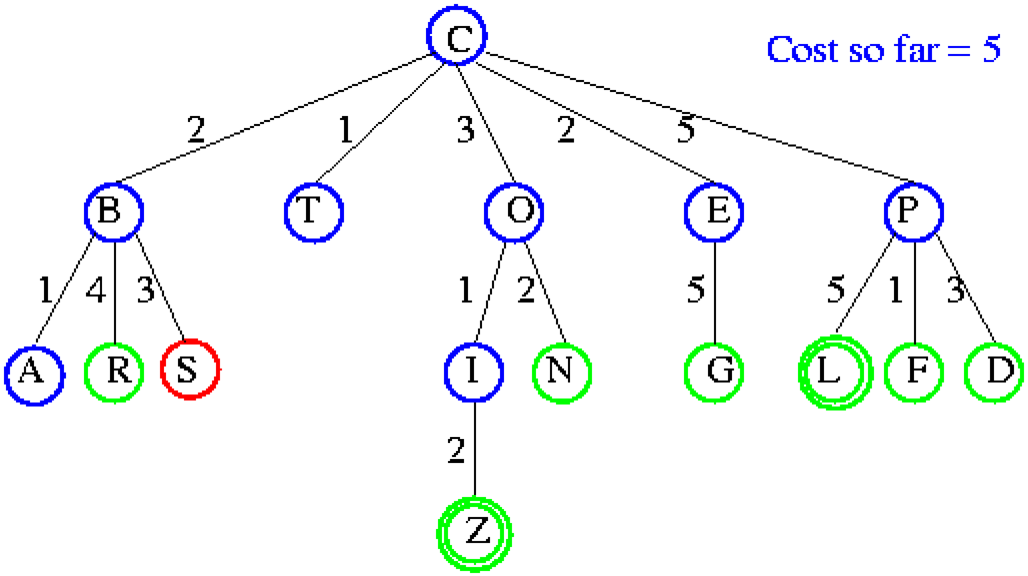
Open list: P(5) S(5) N(5) R(6) Z(6) G(10)

UCS Example



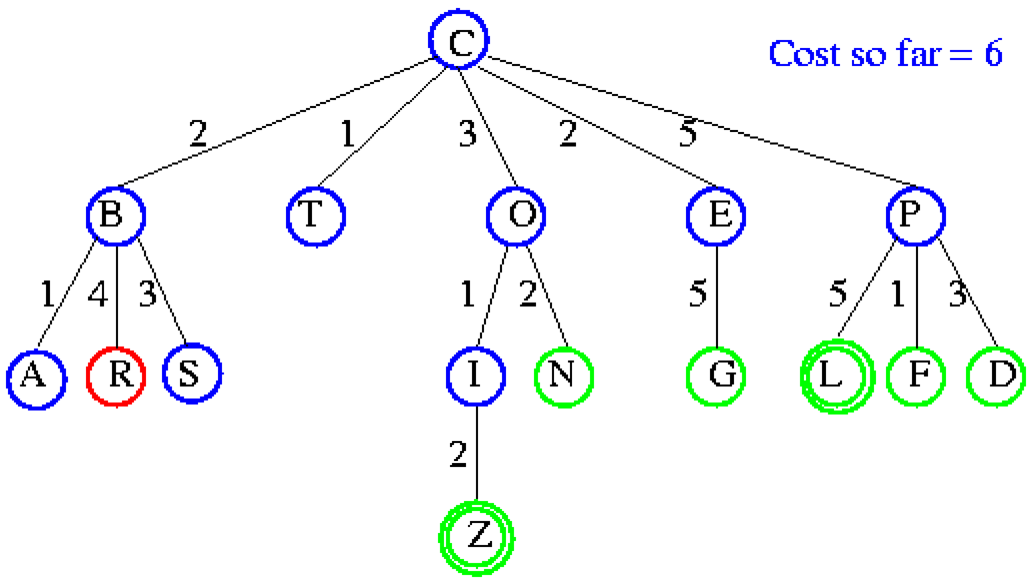
Open list: S(5) N(5) R(6) Z(6) F(6) D(8) G(10) L(10)

UCS Example



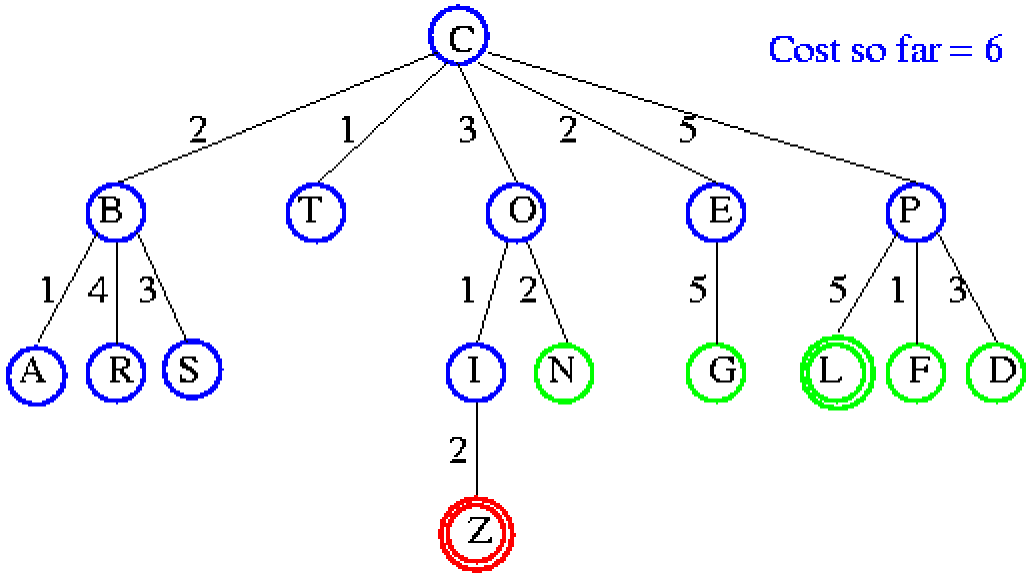
Open list: N(5) R(6) Z(6) F(6) D(8) G(10) L(10)

UCS Example



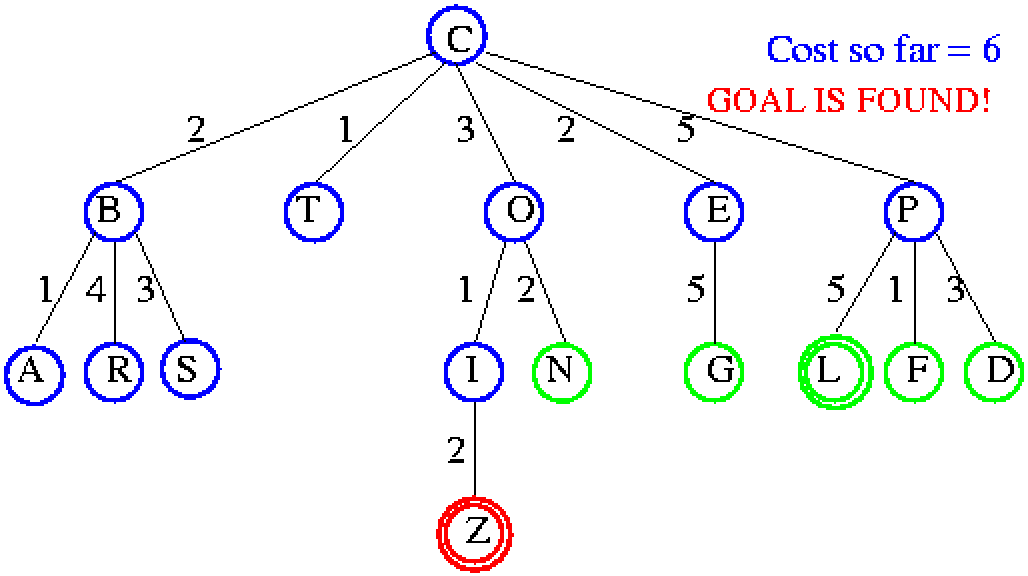
Open list: Z(6) F(6) D(8) G(10) L(10)

UCS Example



Open list: F(6) D(8) G(10) L(10)

UCS Example



C
Cost so far = 6
GOAL IS FOUND!
2
1
3
2
5
B
T
O
E
P
1
4
3
1
2
5
5
1
3
A
R
S
I
N
G
L
F
D
2
Z

Uniform-cost search

Implementation: *fringe* = queue ordered by path cost
Equivalent to breadth-first if all step costs all equal.

Complete? Yes, if step cost ≥ ε
(otherwise it can get stuck in infinite loops)

Time? # of nodes with *path cost* ≤ cost of optimal solution.

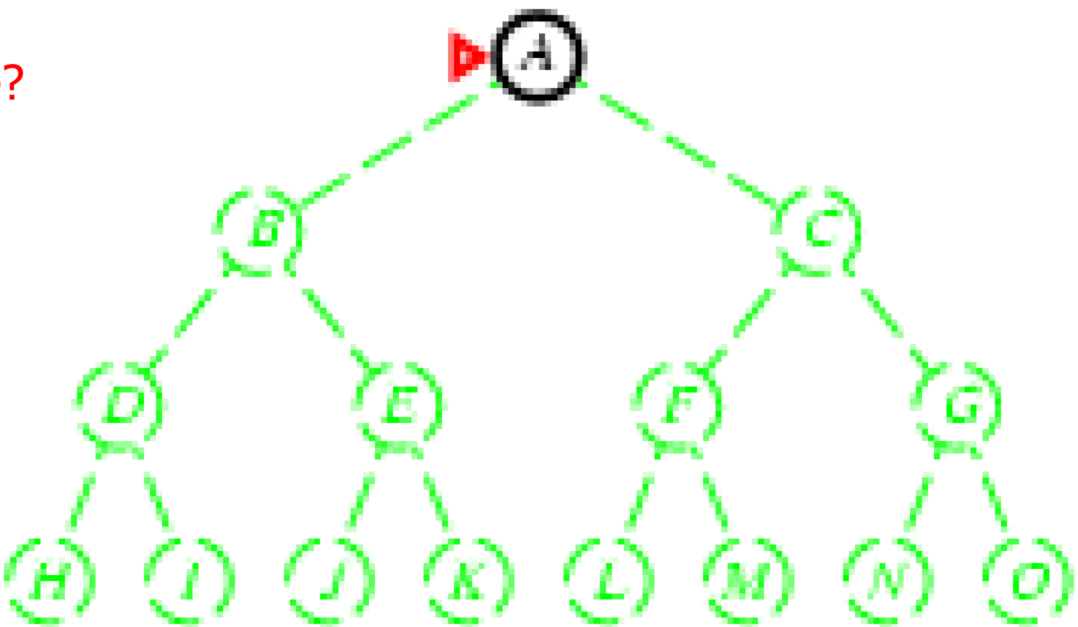
Space? # of nodes on paths with path cost ≤ cost of optimal solution.

Optimal? Yes, for any step cost.

Depth-first search

- Expand *deepest* unexpanded node
- Implementation:
  - *fringe* = Last In First Out (LIPO) queue, i.e., put successors at front

Is A a goal state?

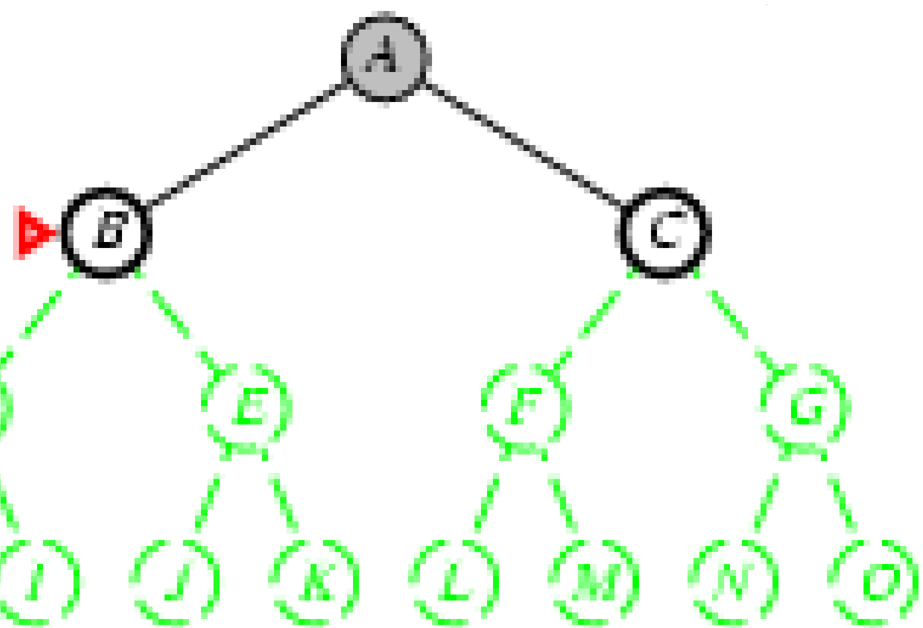


Depth-first search

- Expand deepest unexpanded node
-
- Implementation:
  - *fring*

queue=[B,C]

Is B a goal state?

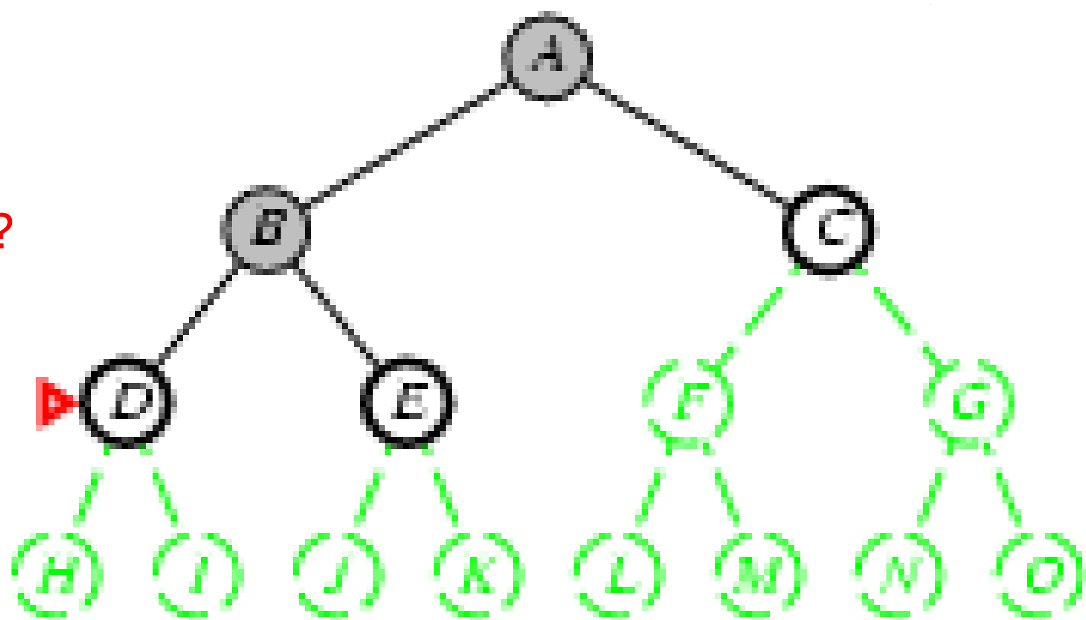


Depth-first search

- Expand deepest unexpanded node
-
- Implementation:
  - *fring*

queue=[D,E,C]

Is D = goal state?

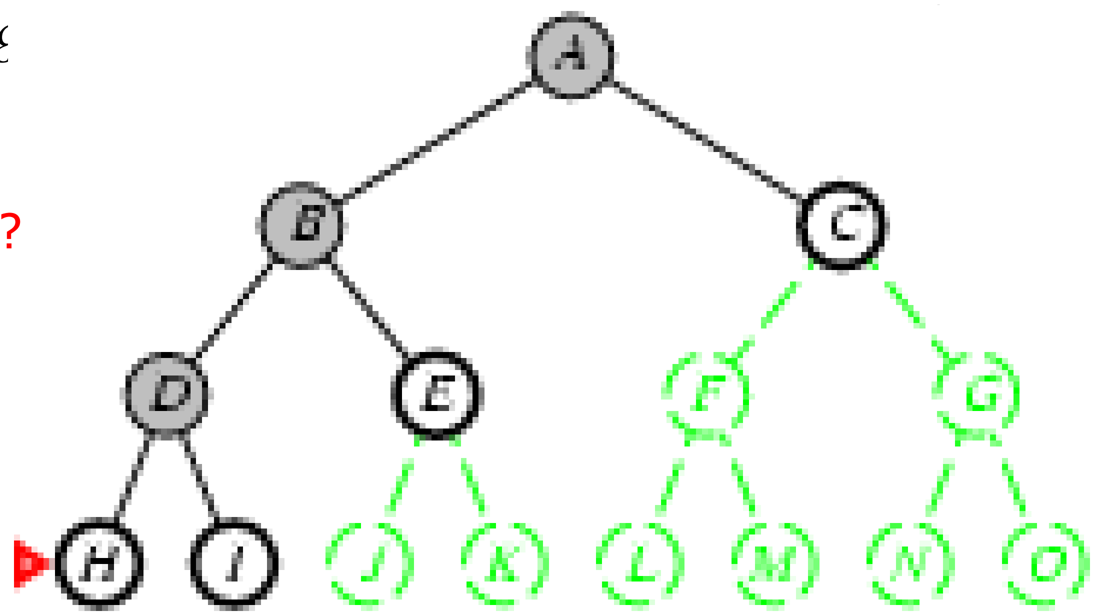


Depth-first search

- Expand deepest unexpanded node
-
- Implementation:
  - *fring*

queue=[H,I,E,C]

Is H = goal state?

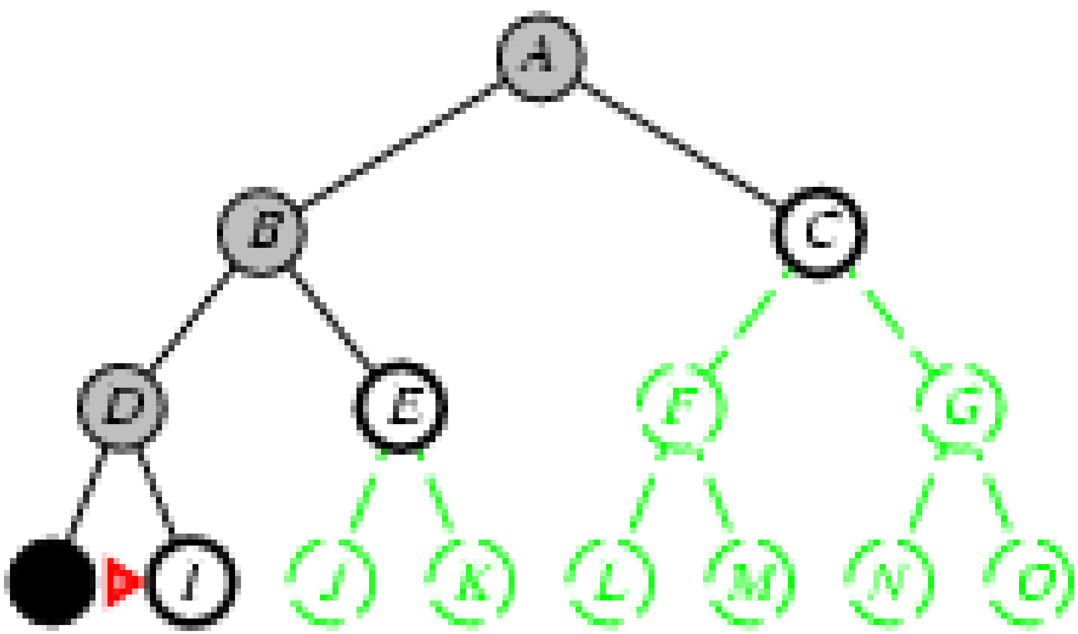


Depth-first search

- Expand deepest unexpanded node
-
- Implementation:
  - *fring*

queue=[I,E,C]

Is I = goal state?

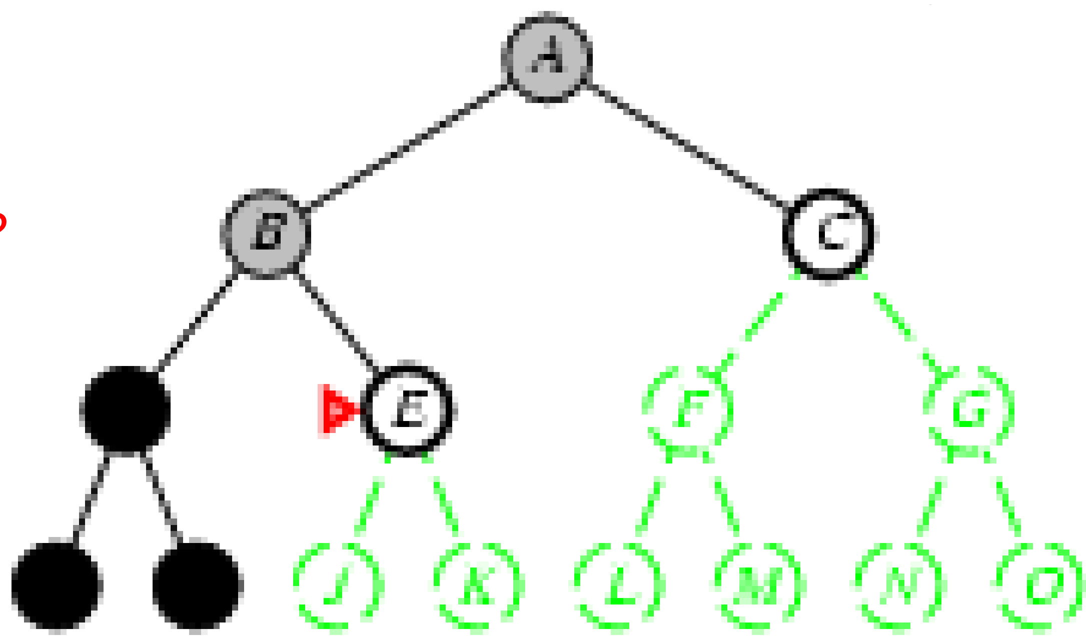


Depth-first search

- Expand deepest unexpanded node
-
- Implementation:
  - *fring*

queue=[E,C]

Is E = goal state?

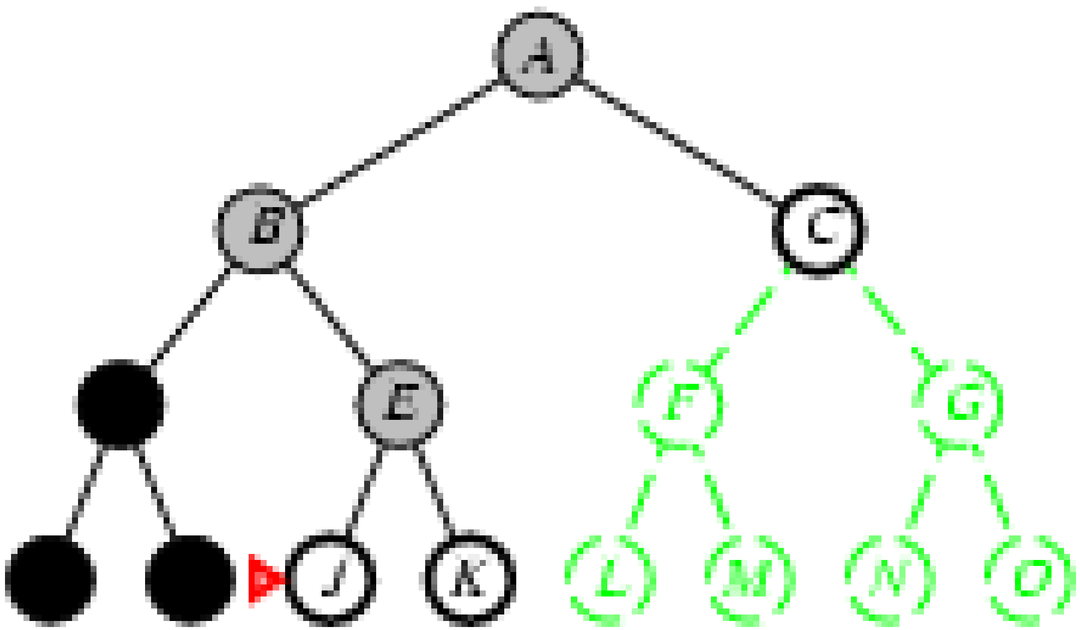


Depth-first search

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

queue=[J,K,C]

Is J = goal state?

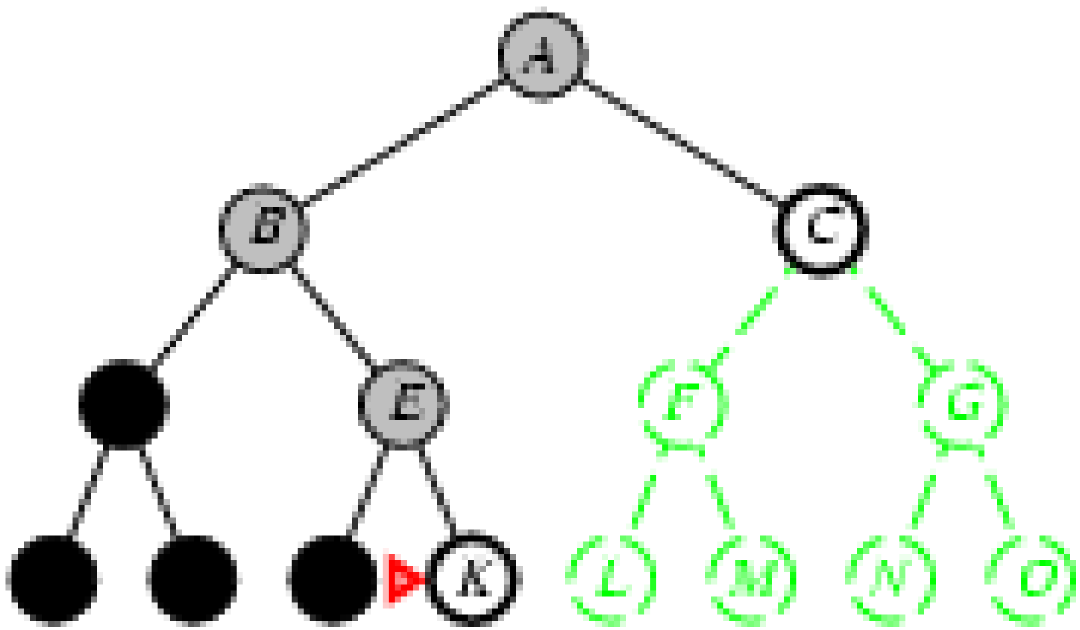


Depth-first search

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

queue=[K,C]

Is K = goal state?

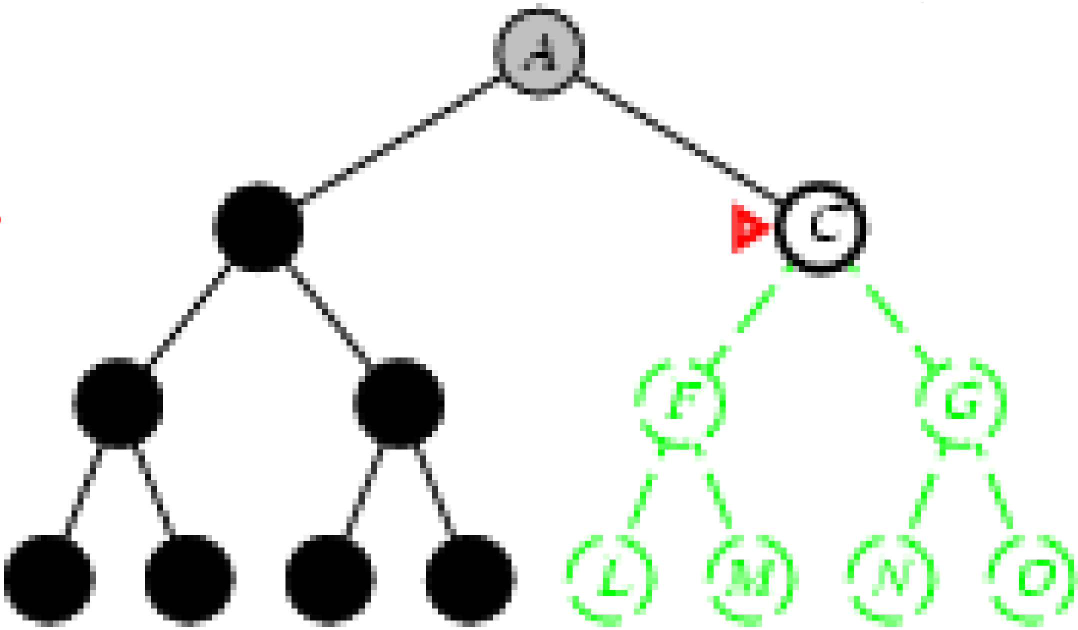


Depth-first search

- Expand deepest unexpanded node
-
- Implementation:
  - *fring*

queue=[C]•

Is C = goal state?

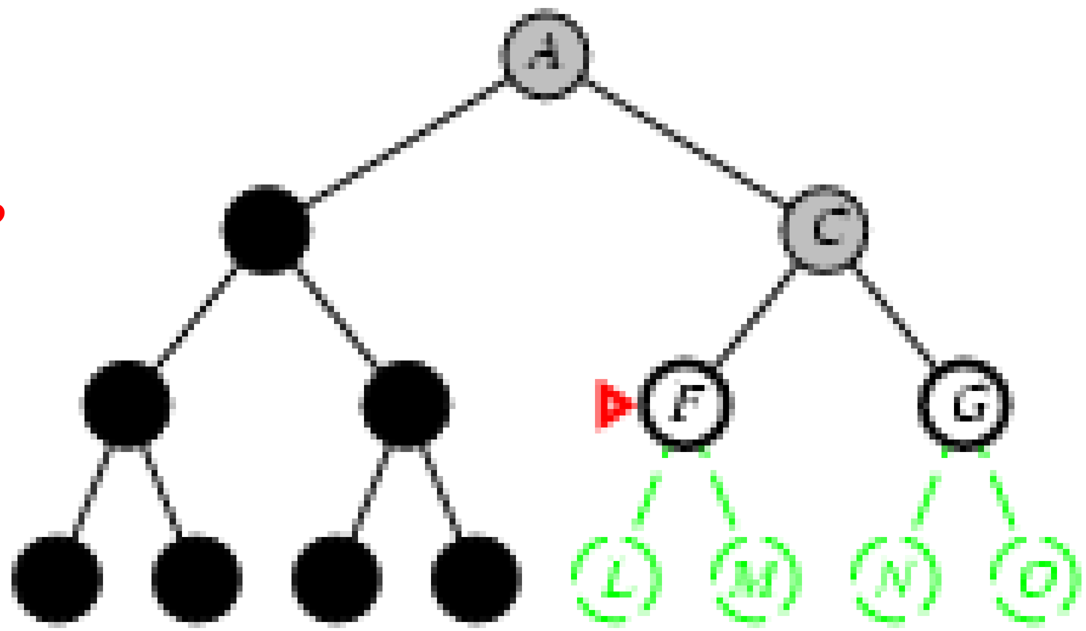


Depth-first search

- Expand deepest unexpanded node
-
- Implementation:
  - *fring*

queue=[F,G]

Is F = goal state?

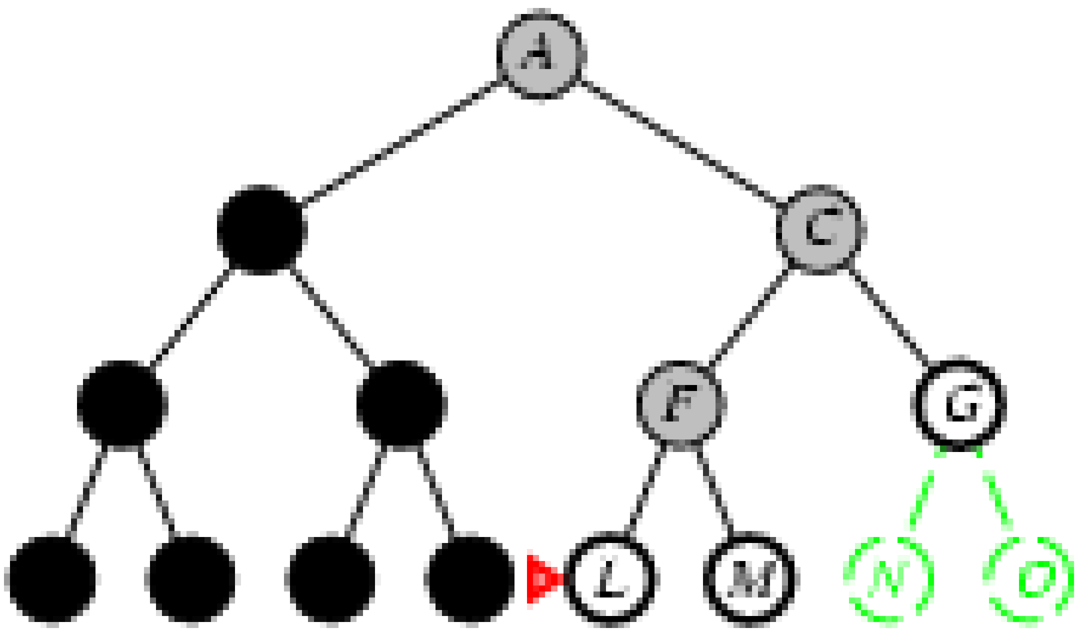


Depth-first search

- Expand deepest unexpanded node
-
- Implementation:
  - *fring*

queue=[L,M,G]

Is L = goal state?

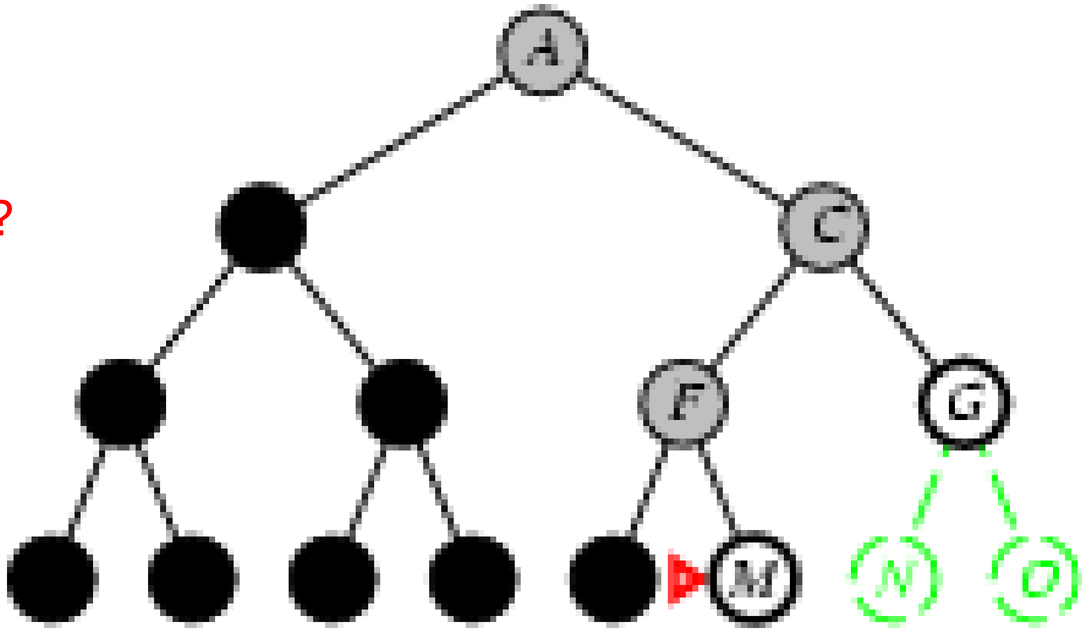


Depth-first search

- Expand deepest unexpanded node
-
- Implementation:
  - *fring*

queue=[M,G]

Is M = goal state?



Properties of depth-first search

- Complete? No: fails in infinite-depth spaces

  Can modify to avoid repeated states along path
- Time? $O(b^m)$ with m=maximum depth
- 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! (we only need to remember a single path + expanded unexplored nodes)
- Optimal? No (It may find a non-optimal goal first)

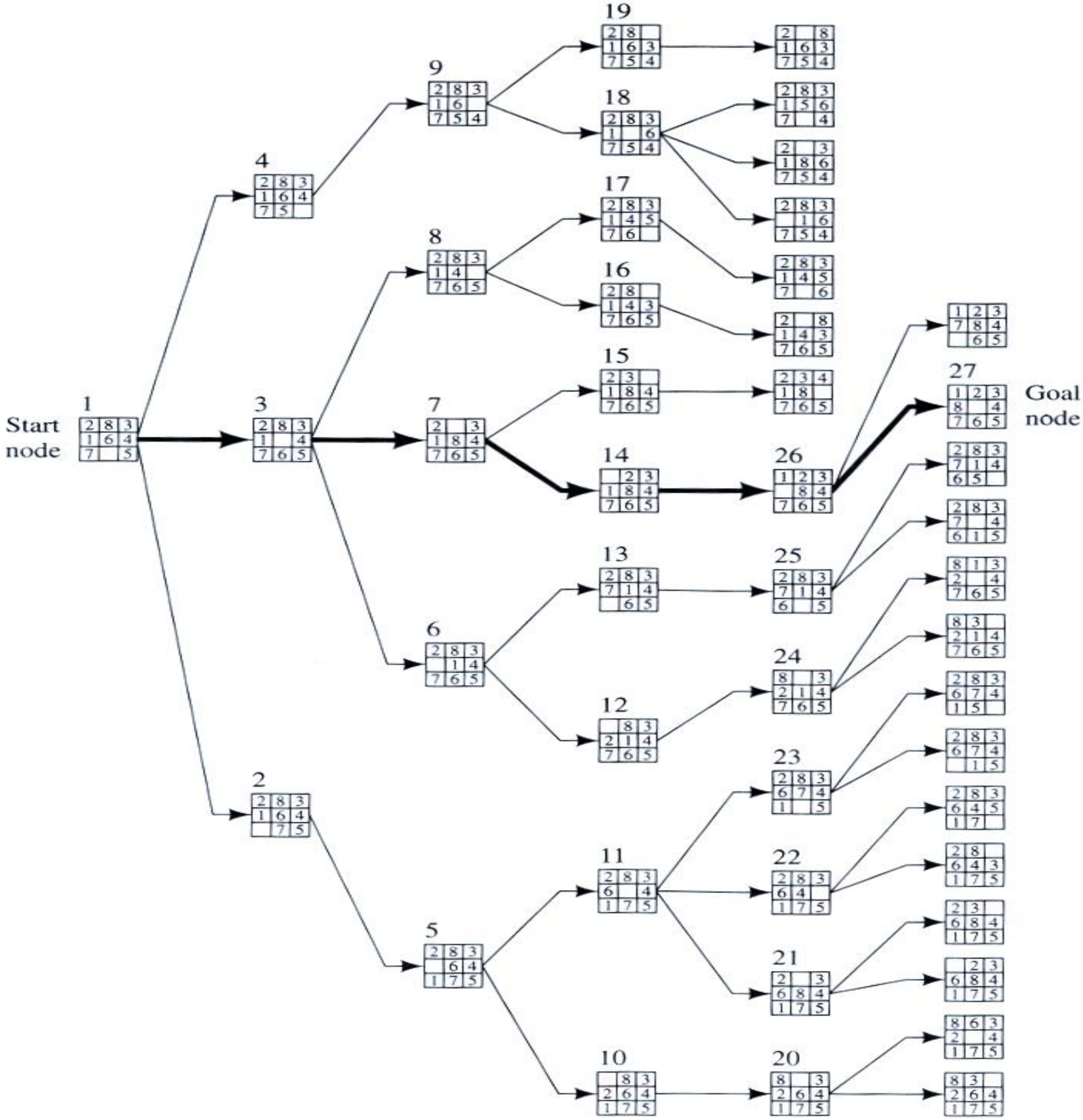
8 Puzzel Game

2	8	3
1	6	4
7		5

8		3
2	6	4
1	7	5

Initial State	Goal State

Example BFS


Start node
Goal node
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27

Iterative deepening search

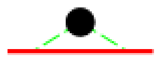
- To avoid the infinite depth problem of DFS, we can decide to only search until depth L, i.e. we don't expand beyond depth L. → Depth-Limited Search

- What of solution is deeper than L? → Increase L iteratively. → Iterative Deepening Search

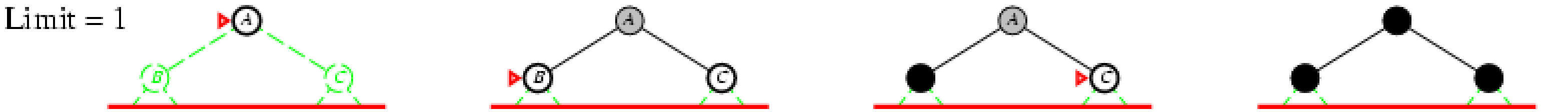
- As we shall see: this inherits the memory advantage of Depth-First search.

Iterative deepening search L=0

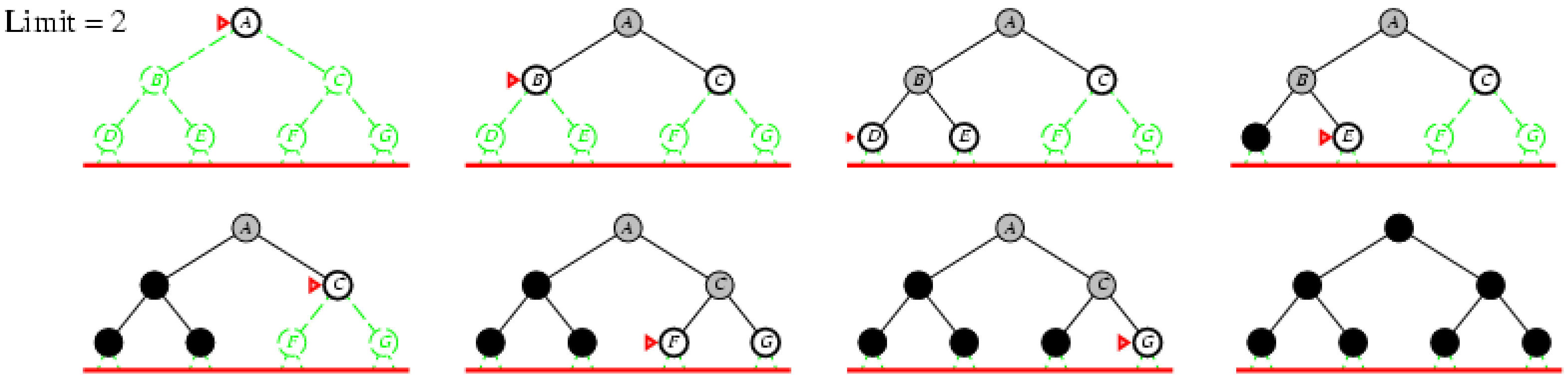
Limit = 0



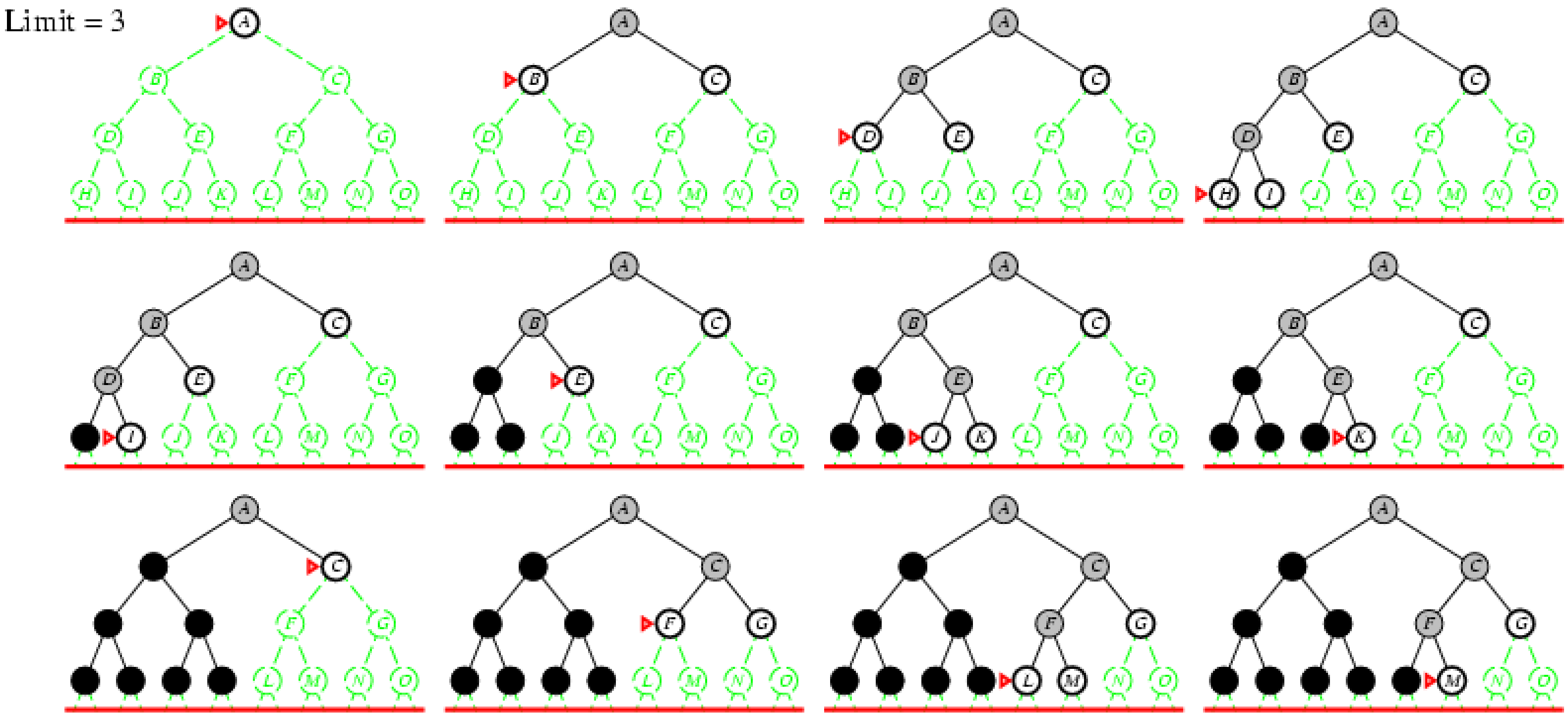
Iterative deepening search *L*=1



Iterative deepening search *l*L=2



Iterative deepening search *l*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*:

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

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

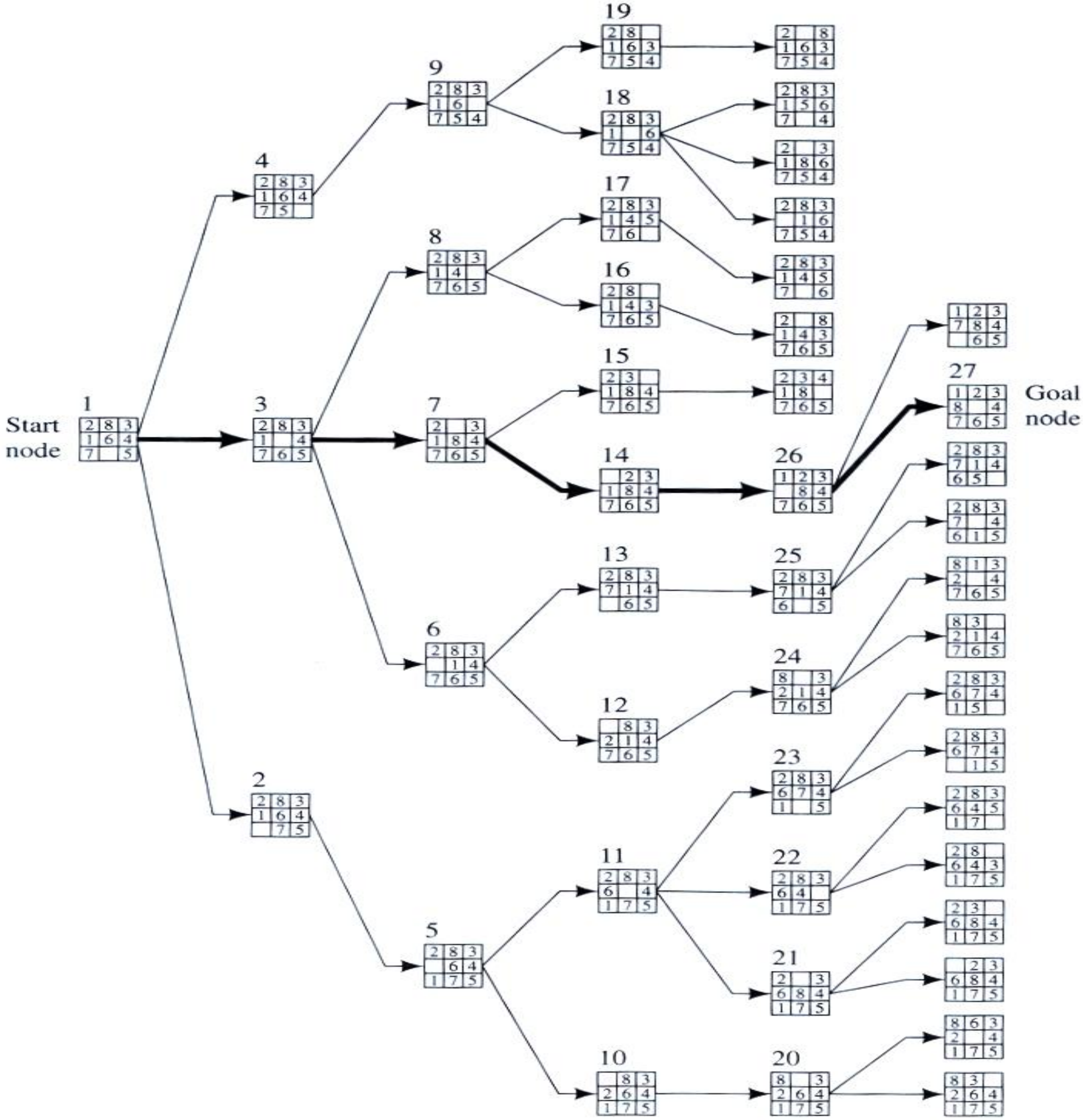
BFS

- For $b = 10$, $d = 5$,
  - $N_{DLS} = 1 + 10 + 100 + 1{,}000 + 10{,}000 + 100{,}000 = 111{,}111$
  - $N_{IDS} = 6 + 50 + 400 + 3{,}000 + 20{,}000 + 100{,}000 = 123{,}450$
  - N_{BFS} = ... $= 1{,}111{,}100$

Properties of iterative deepening search

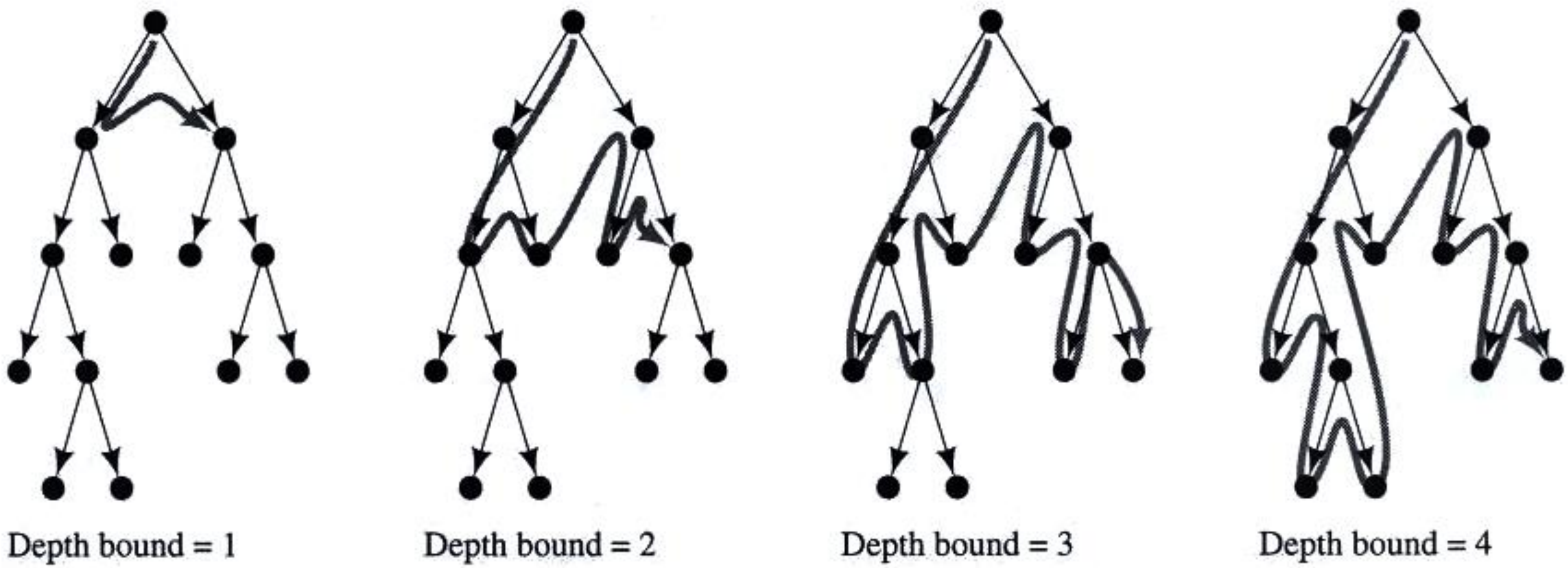
- Complete? Yes
- Time? $(d+1)b^0 + d\,b^1 + (d-1)b^2 + \ldots + b^d = O(b^d)$
- Space? $O(bd)$
- Optimal? Yes, if step cost = 1 or increasing function of depth.

Example BFS



Start node
Goal node
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27

Example IDS



Stages in Iterative-Deepening Search

Bidirectional Search

- Idea
  - simultaneously search forward from S and backwards from G
  - stop when both "meet in the middle"
  - need to keep track of the intersection of 2 open sets of nodes
- What does searching backwards from G mean
  - need a way to specify the predecessors of G
    - this can be difficult,
    - e.g., predecessors of checkmate in chess?
  - what if there are multiple goal states?
  - what if there is only a goal test, no explicit list?

Bi-Directional Search

Complexity: time and space complexity are: $O(b^{d/2})$

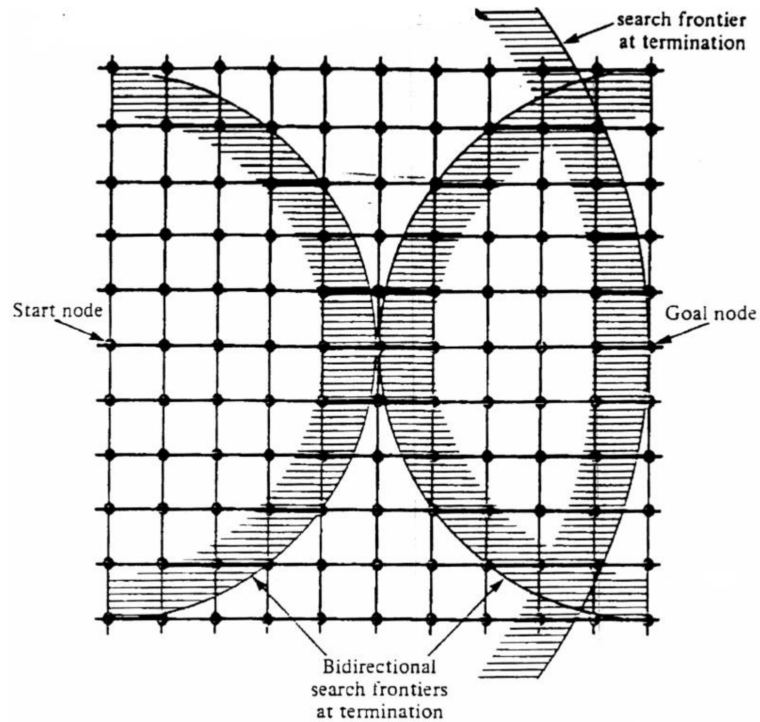


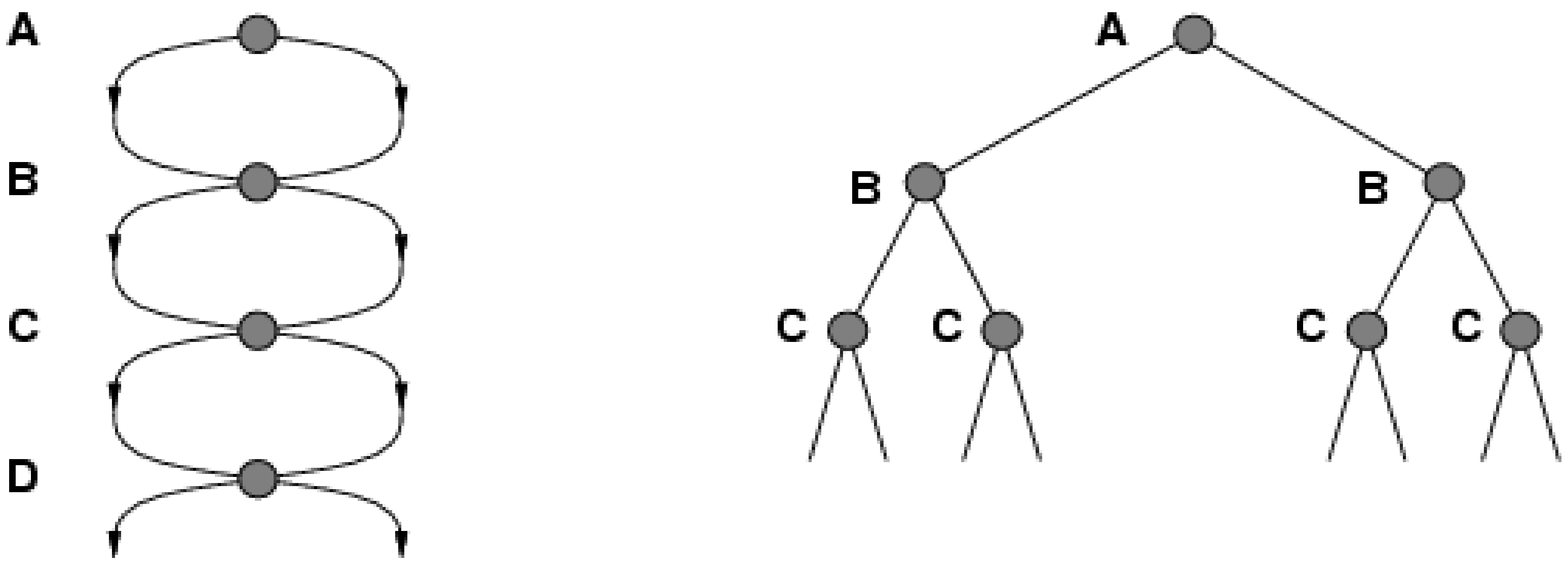
Fig. 2.10 Bidirectional and unidirectional breadth-first searches.

Summary of algorithms

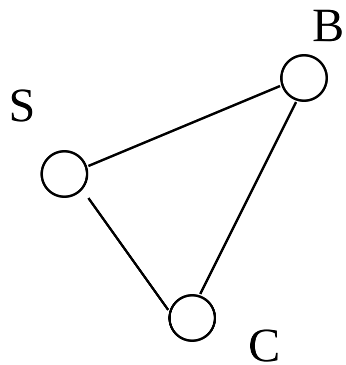
Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening
Complete?	Yes	Yes	No	No	Yes
Time	$O(b^{d+1})$	$O(b^{\lceil C^*/\epsilon \rceil})$	$O(b^m)$	$O(b^l)$	$O(b^d)$
Space	$O(b^{d+1})$	$O(b^{\lceil C^*/\epsilon \rceil})$	$O(bm)$	$O(bl)$	$O(bd)$
Optimal?	Yes	Yes	No	No	Yes

Repeated states

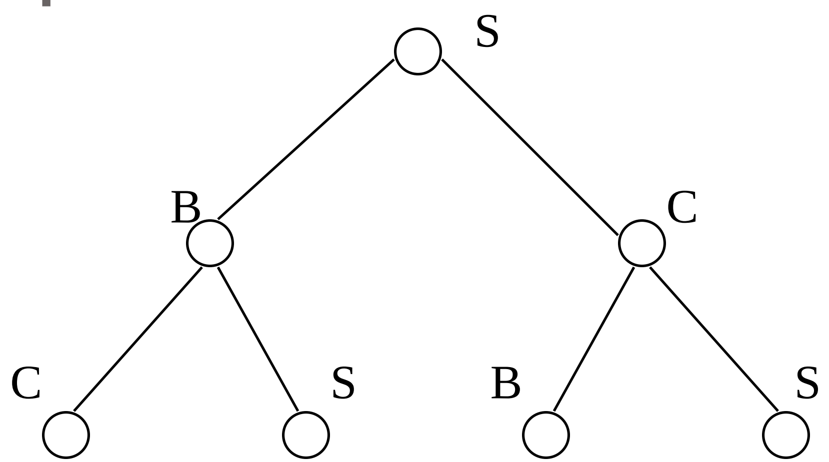
- Failure to detect repeated states can turn a linear problem into an exponential one!



Solutions to Repeated States



State Space



Example of a Search Tree

- Method 1 ← suboptimal but practical
  - do not create paths containing cycles (loops)
- Method 2 ← optimal but memory inefficient
  - never generate a state generated before
    - must keep track of all possible states (uses a lot of memory)
    - e.g., 8-puzzle problem, we have 9! = 362,880 states