Artificial Intelligence

K. N Toosi University of Technology

Course Instructor:
Dr. Omid Azarkasb
Teaching Assistants:
Atena Najaf Abadi Farahani
Saeed Mahmoudian

Uninformed Search

Search

- Search permeates all of AI
- What choices are we searching through?
  - Problem solving
    Action combinations (move 1, then move 3, then move 2...)
  - Natural language
    Ways to map words to parts of speech
  - Computer vision
    Ways to map features to object model
  - Machine learning
    Possible concepts that fit examples seen so far
  - Motion planning
    Sequence of moves to reach goal destination
- An intelligent agent is trying to find a set or sequence of actions to achieve a goal
- This is a goal-based agent

Simple Problem Solving Agent

function SIMPLE-PROBLEM-SOLVING-AGENT(*percept*) **return** an action
    static: *seq* , an action sequence
        state, some description of the current world state
        goal, a goal
        problem, a problem formulation

    state ← UPDATE-STATE(*state*, *percept*)
    if *seq* is empty **then**
        goal ← FORMULATE-GOAL(*state*)
        problem ← FORMULATE-PROBLEM(*state* , *goal*)
        seq ← SEARCH(*problem*)
    action ← FIRST(*seq*)
    seq ← REST(*seq*)
    return *action*

Problem-solving Agent

```
SimpleProblemSolvingAgent(percept)
  state = UpdateState(state, percept)
  if sequence is empty then
  goal = FormulateGoal(state)
  problem = FormulateProblem(state, g)
  sequence = Search(problem)
  action = First(sequence)
  sequence = Rest(sequence)
  Return action
```

Assumptions

- Static or dynamic?

Environment is static

Assumptions

- Static or dynamic?
- Fully or partially observable?

Environment is fully observable

Assumptions

- Static or dynamic?
- Fully or partially observable?
- Discrete or continuous?

Environment is discrete

Assumptions

- Static or dynamic?
- Fully or partially observable?
- Discrete or continuous?
- Deterministic or stochastic?

Environment is deterministic

Assumptions

- Static or dynamic?
- Fully or partially observable?
- Discrete or continuous?
- Deterministic or stochastic?
- Episodic or sequential?

Environment is sequential

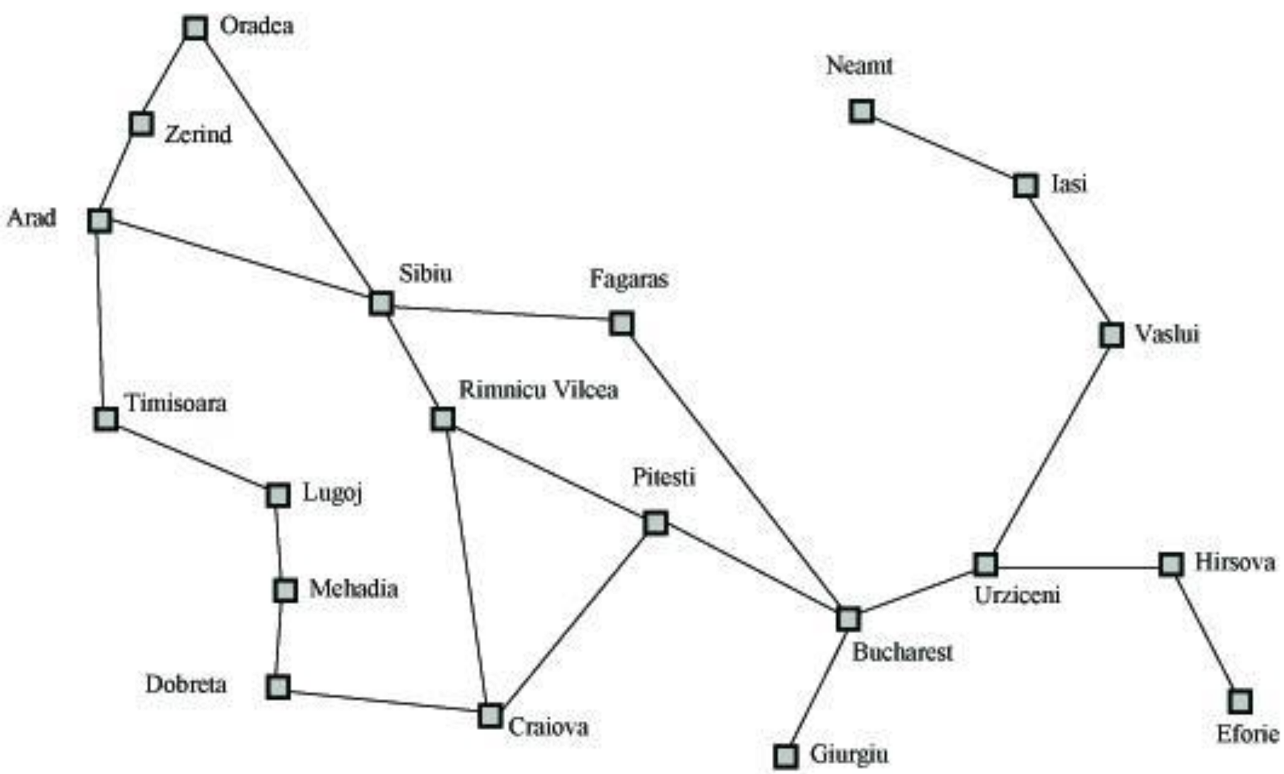
Assumptions

- Static or dynamic?
- Fully or partially observable?
- Discrete or continuous?
- Deterministic or stochastic?
- Episodic or sequential?
- Single agent or multiple agent?

Assumptions

- Static or dynamic?
- Fully or partially observable?
- Discrete or continuous?
- Deterministic or stochastic?
- Episodic or sequential?
- Single agent or multiple agent?

Search Example



Formulate goal: Be in Bucharest.

Formulate problem: states are cities, operators drive between pairs of cities

Find solution: Find a sequence of cities (e.g., Arad, Sibiu, Fagaras, Bucharest) that leads from the current state to a state meeting the goal condition

Search Space Definitions

- State
  - A description of a possible state of the world
  - Includes all features of the world that are pertinent to the problem
- Initial state
  - Description of all pertinent aspects of the state in which the agent starts the search
- Goal test
  - Conditions the agent is trying to meet (e.g., have $1M)
- Goal state
  - Any state which meets the goal condition
  - Thursday, have $1M, live in NYC
  - Friday, have $1M, live in Valparaiso
- Action
  - Function that maps (transitions) from one state to another

Search Space Definitions

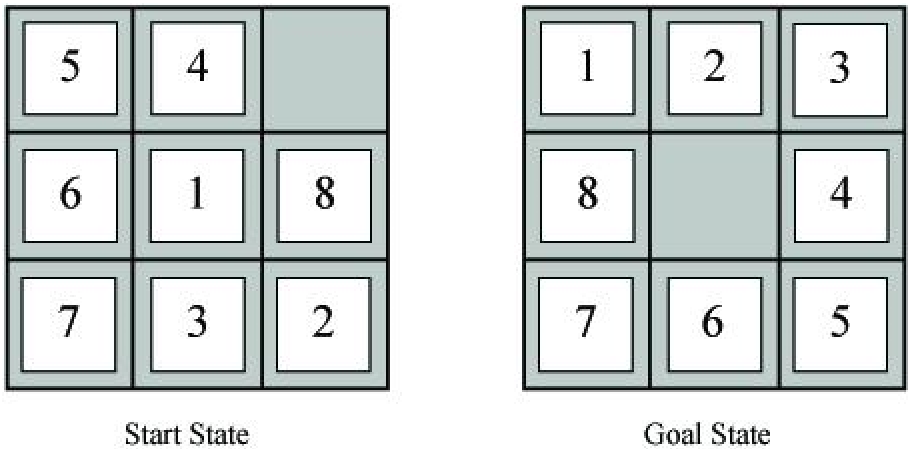
- Problem formulation
  - Describe a general problem as a search problem
- Solution
  - Sequence of actions that transitions the world from the initial state to a goal state
- Solution cost (additive)
  - Sum of the cost of operators
  - Alternative: sum of distances, number of steps, etc.
- Search
  - Process of looking for a solution
  - Search algorithm takes problem as input and returns solution
  - We are searching through a space of possible states
- Execution
  - Process of executing sequence of actions (solution)

Problem Formulation

A search problem is defined by the

1. Initial state (e.g., Arad)
2. Operators (e.g., Arad -> Zerind, Arad -> Sibiu, etc.)
3. Goal test (e.g., at Bucharest)
4. Solution cost (e.g., path cost)

Example Problems – Eight Puzzle



States: tile locations

Initial state: one specific tile configuration

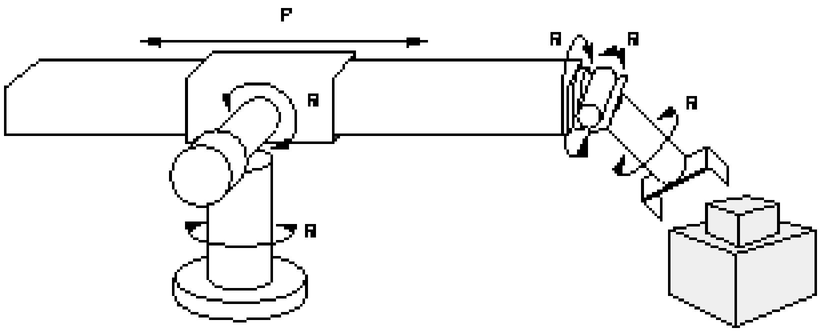
Operators: move blank tile left, right, up, or down

Goal: tiles are numbered from one to eight around the square

Path cost: cost of 1 per move (solution cost same as number of most or path length)

Eight puzzle applet

Example Problems – Robot Assembly



States: real-valued coordinates of

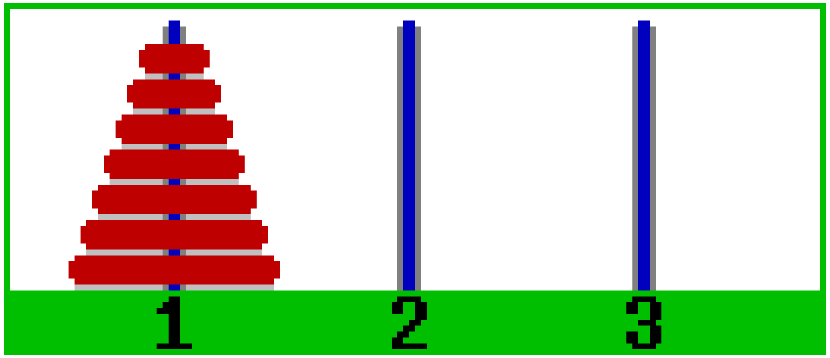
- robot joint angles
- parts of the object to be assembled

Operators: rotation of joint angles

Goal test: complete assembly

Path cost: time to complete assembly

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

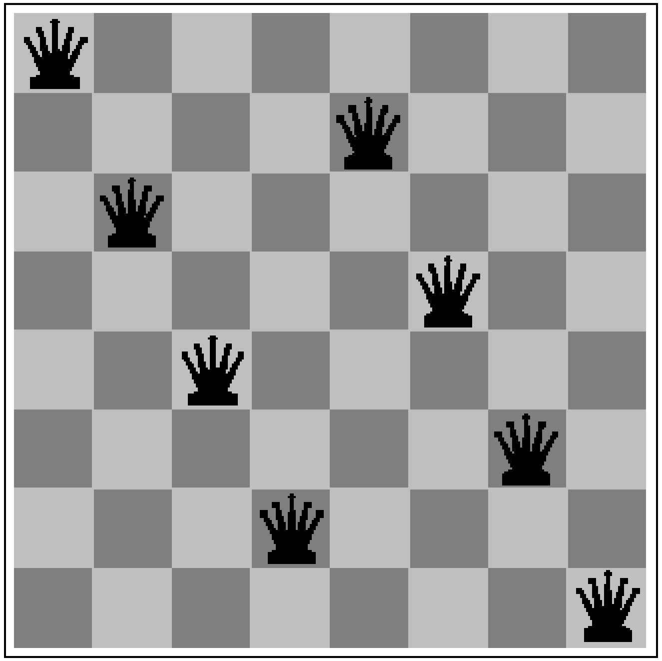
Operators: rotate row x or column y on face z direction a

Goal: configuration has only one color on each face

Path cost: 1 per move

Rubik's cube applet

Example Problems – Eight Queens



States: locations of 8 queens on chess board

Initial state: one specific queens configuration

Operators: move queen x to row y and column z

Goal: no queen can attack another (cannot be in same row, column, or diagonal)

Path cost: 0 per move

Eight queens applet

Example Problems – Missionaries and Cannibals

States: number of missionaries, cannibals, and boat on near river bank

Initial state: all objects on near river bank

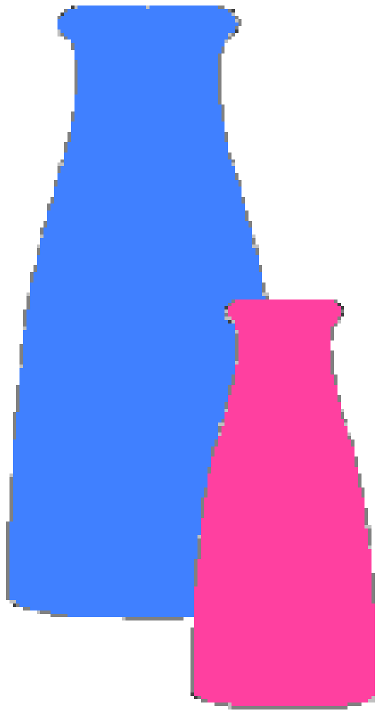
Operators: move boat with x missionaries and y cannibals to other side of river
- no more cannibals than missionaries on either river bank or in boat
- boat holds at most m occupants

Goal: all objects on far river bank

Path cost: 1 per river crossing

Missionaries and cannibals applet

Example Problems –Water Jug



States: Contents of 4-gallon jug and 3-gallon jug

Initial state: (0,0)

Operators:
- fill jug x from faucet
- pour contents of jug x in jug y until y full
- dump contents of jug x down drain

Goal: (2,n)

Path cost: 1 per fill

Saving the world, Part I

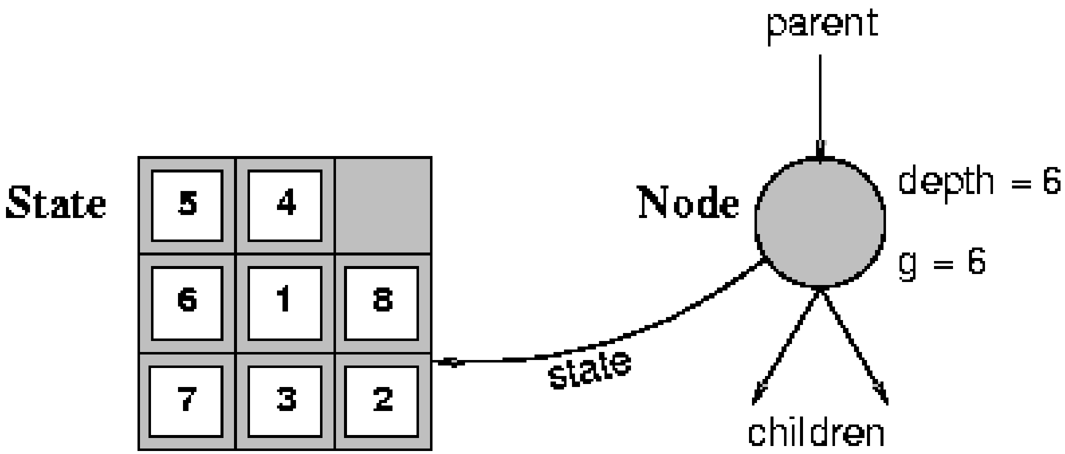
Saving the world, Part II

Sample Search Problems

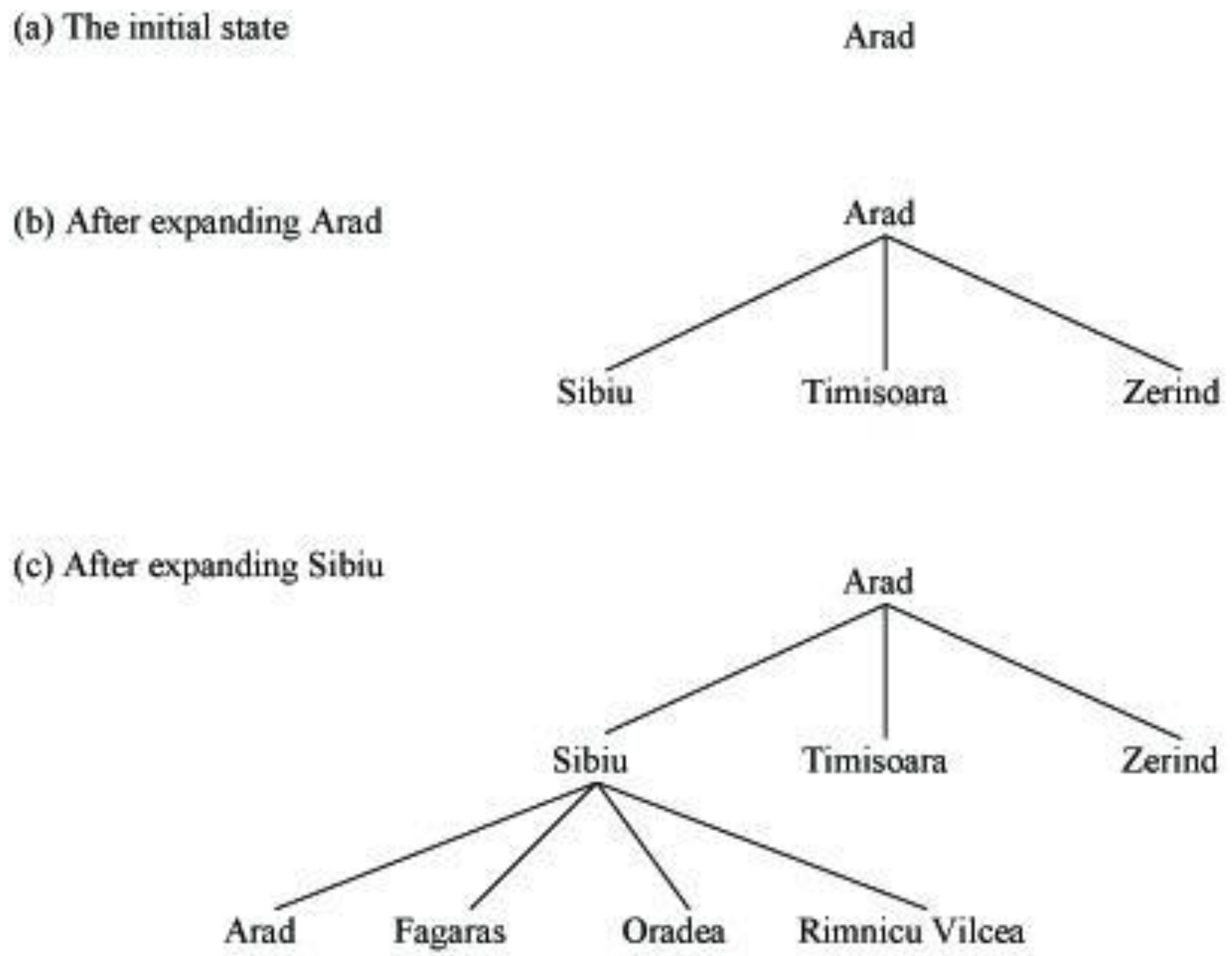
- Graph coloring
- Protein folding
- Game playing
- Airline travel
- Proving algebraic equalities
- Robot motion planning

Visualize Search Space as a Tree

- States are nodes
- Actions are edges
- Initial state is root
- Solution is path from root to goal node
- Edges sometimes have associated costs
- States resulting from operator are children



Search Problem Example (as a tree)



Search Function – Uninformed Searches

```
Open = initial state                    // open list is all generated states
                                        // that have not been "expanded"
While open not empty                    // one iteration of search algorithm
  state = First(open)                   // current state is first state in open
  Pop(open)                             // remove new current state from open
  if Goal(state)                        // test current state for goal condition
    return "succeed"                    // search is complete
                                        // else expand the current state by
                                        // generating children and
                                        // reorder open list per search strategy
  else open = QueueFunction(open, Expand(state))
Return "fail"
```

Tree Search Algorithm

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

Tree Search Algorithm

```
function TREE-SEARCH(problem, fringe) return a solution or failure
    fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
    loop do
        if EMPTY?(fringe) then return failure
        node ← REMOVE-FIRST(fringe)
        if      GOAL-TEST[node] then
                 return SOLUTION(node)
        else
                fringe ← INSERT-ALL(EXPAND(node, problem), fringe)
```

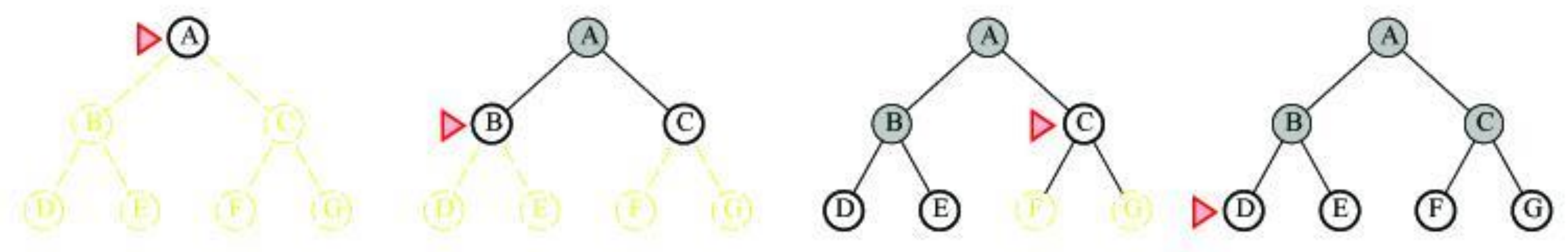
Search Strategies

- Search strategies differ only in QueuingFunction
- Features by which to compare search strategies
  - Completeness (always find solution)
  - Cost of search (time and space)
  - Cost of solution, optimal solution
  - Make use of knowledge of the domain
    - “uninformed search” vs. “informed search”

Breadth-First Search

- Generate children of a state, QueueingFn adds the children to the end of the open list
- Level-by-level search
- Order in which children are inserted on open list is arbitrary
- In tree, assume children are considered left-to-right unless specified differently
- Number of children is “branching factor” b

BFS Examples



b = 2

[Example trees]

[Search algorithms applet]

Analysis

- Assume goal node at level d with constant branching factor b

- Time complexity (measured in #nodes generated)
  - 1 (1^{st} level) + b (2^{nd} level) + b^2 (3^{rd} level) + … + b^d (goal level) + (b^{d+1} – b) = O(b^d+1)

- This assumes goal on far right of level
- Space complexity
  - At most majority of nodes at level d + majority of nodes at level d+1 = O(b^{d+1})
  - Exponential time and space

- Features
  - Simple to implement
  - Complete
  - Finds shortest solution (not necessarily least-cost unless all operators have equal cost)

Analysis

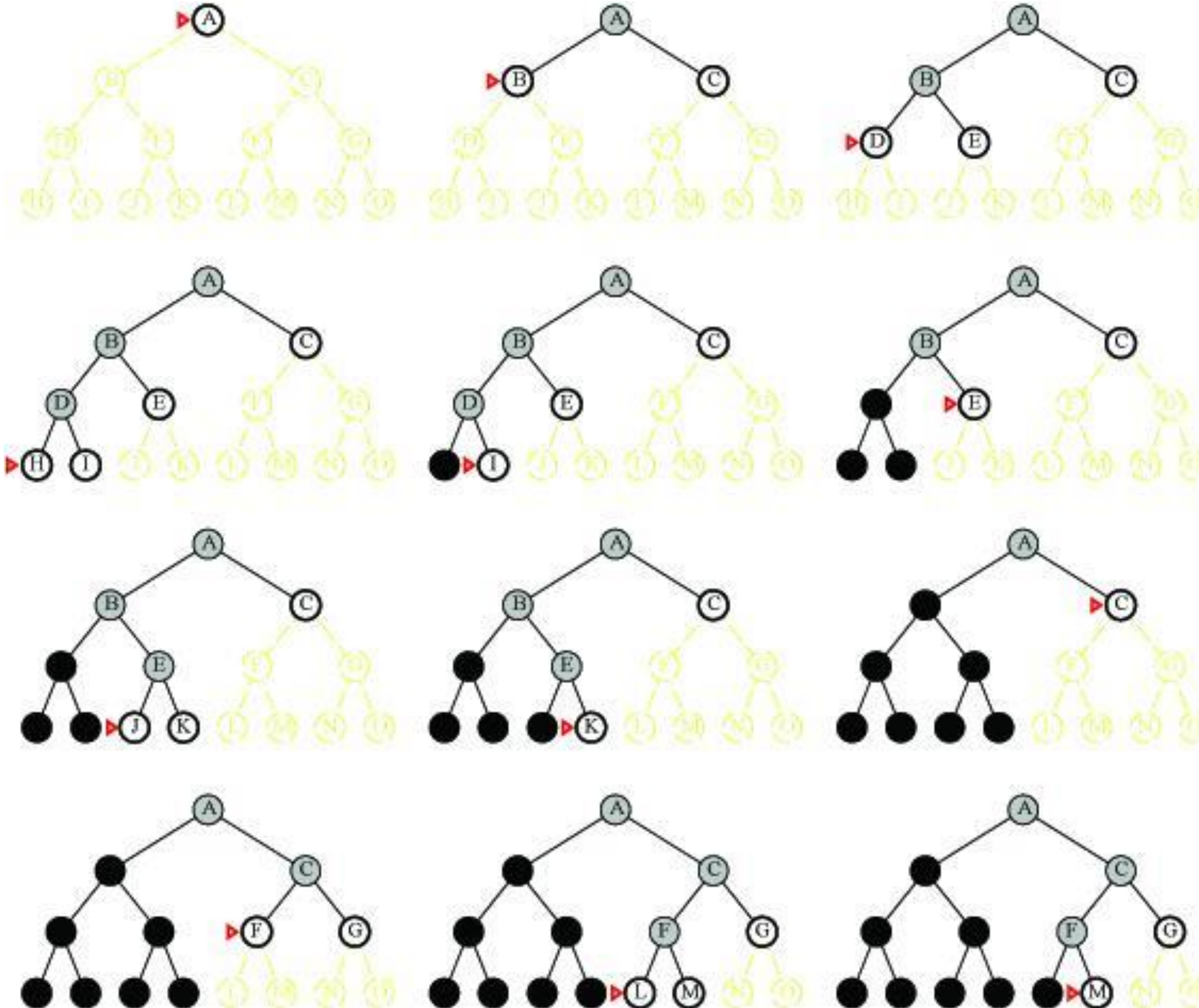
- See what happens with b=10
  - expand 10,000 nodes/second
  - 1,000 bytes/node

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

Depth-First Search

- QueueingFn adds the children to the front of the open list
- BFS emulates FIFO queue
- DFS emulates LIFO stack
- Net effect
  - Follow leftmost path to bottom, then backtrack
  - Expand deepest node first

DFS Examples



Example trees

Analysis

- Time complexity
  - In the worst case, search entire space
  - Goal may be at level d but tree may continue to level m, m>=d
  - $O(b^m)$
  - Particularly bad if tree is infinitely deep

- Space complexity
  - Only need to save one set of children at each level
  - 1 + b + b + … + b (m levels total) = O(bm)
  - For previous example, DFS requires 118kb instead of 10 petabytes for d=12 (10 billion times less)

- Benefits
  - May not always find solution
  - Solution is not necessarily shortest or least cost
  - If many solutions, may find one quickly (quickly moves to depth d)
  - Simple to implement
  - Space often bigger constraint, so more usable than BFS for large problems

Comparison of Search Techniques

	DFS	BFS
Complete	N	Y
Optimal	N	N
Heuristic	N	N
Time	b^m	b^{d+1}
Space	bm	b^{d+1}

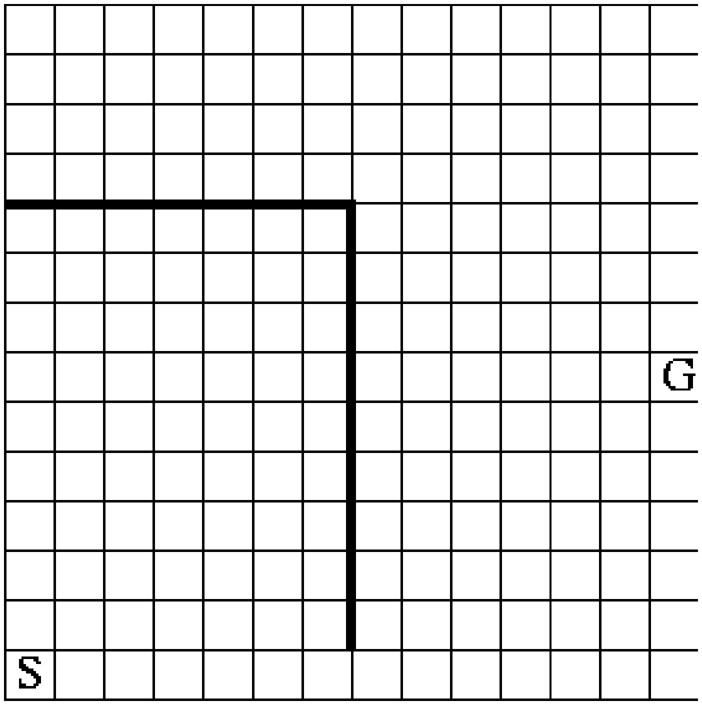
Avoiding Repeated States

Can we do it?

- Do not return to parent or grandparent state
  - In 8 puzzle, do not move up right after down
- Do not create solution paths with cycles
- Do not generate repeated states (need to store and check potentially large number of states)

Maze Example

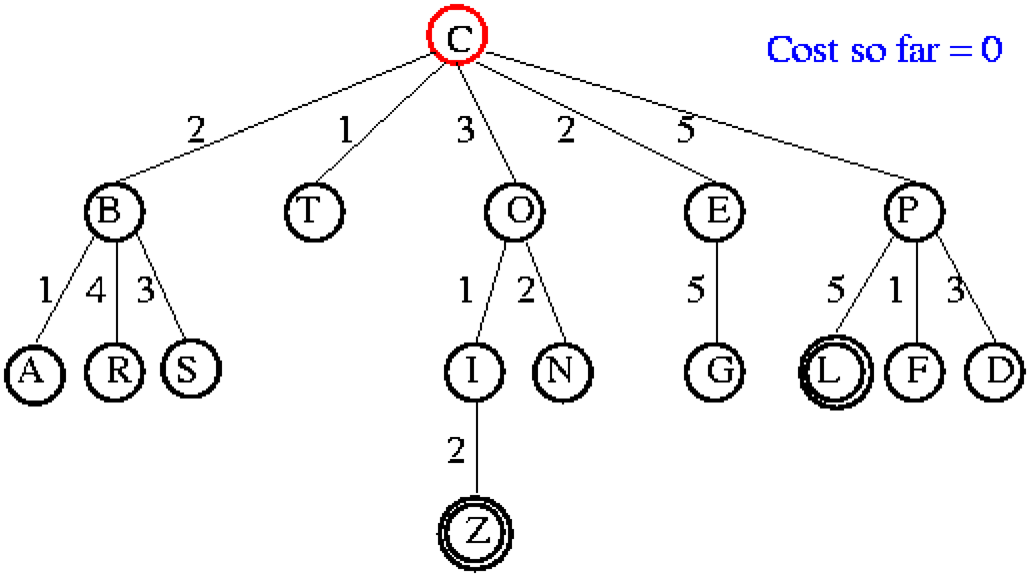
- States are cells in a maze
- Move N, E, S, or W
- What would BFS do (expand E, then N, W, S)?
- What would DFS do?
- What if order changed to N, E, S, W and loops are prevented?



Uniform Cost Search (Branch&Bound)

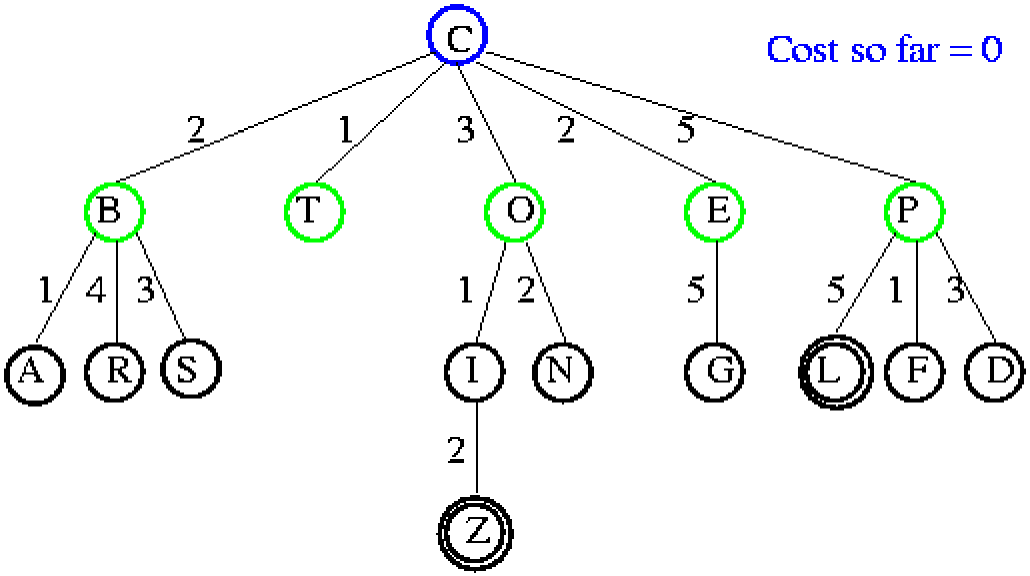
- QueueingFn is SortByCostSoFar
- Cost from root to current node n is g(n)
  - Add operator costs along path
- First goal found is least-cost solution
- Space & time can be exponential because large subtrees with inexpensive steps may be explored before useful paths with costly steps
- If costs are equal, time and space are $O(b^d)$
  - Otherwise, complexity related to cost of optimal solution

UCS Example



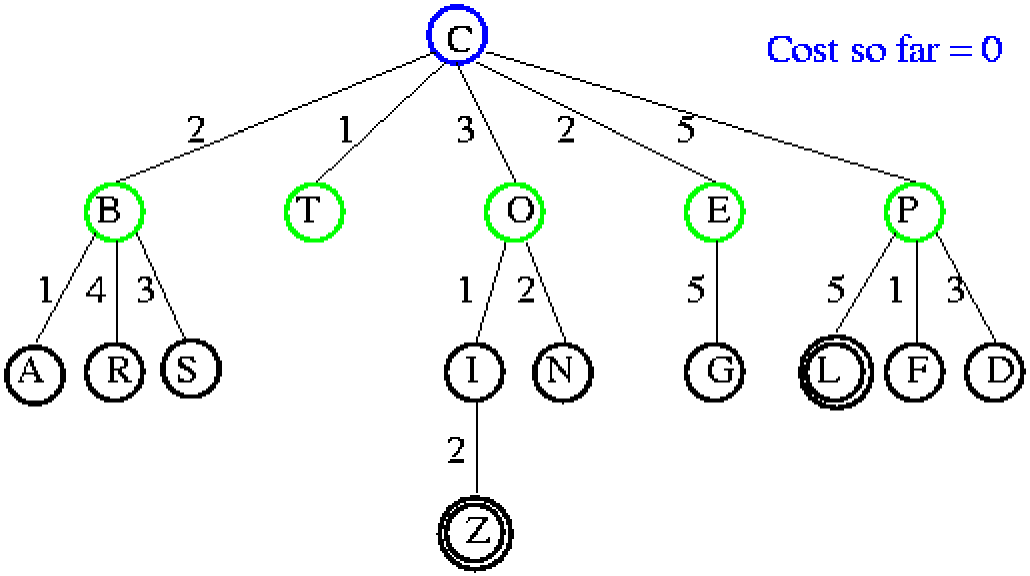
Open list: C

UCS Example



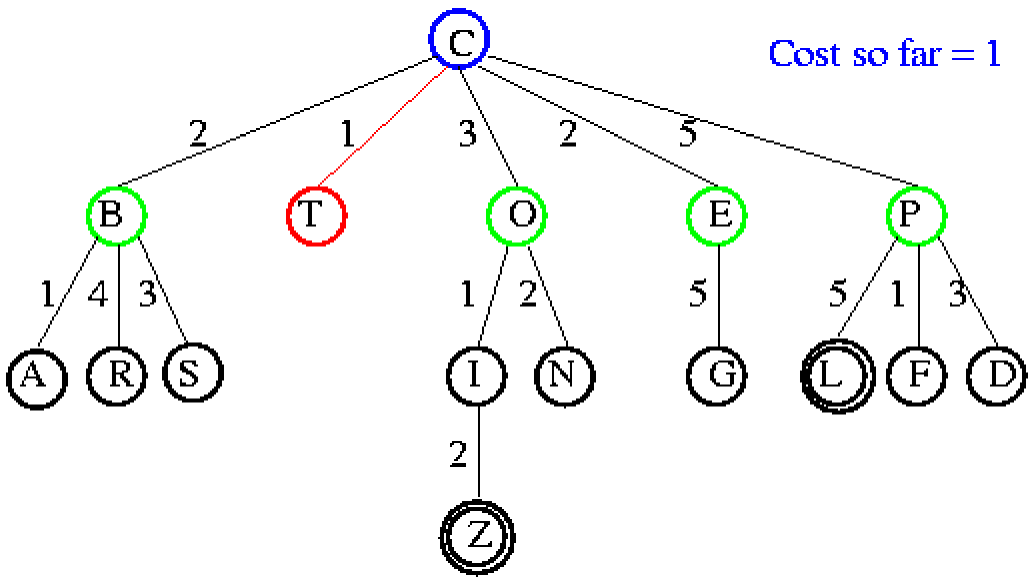
Open list: B(2) T(1) O(3) E(2) P(5)

UCS Example



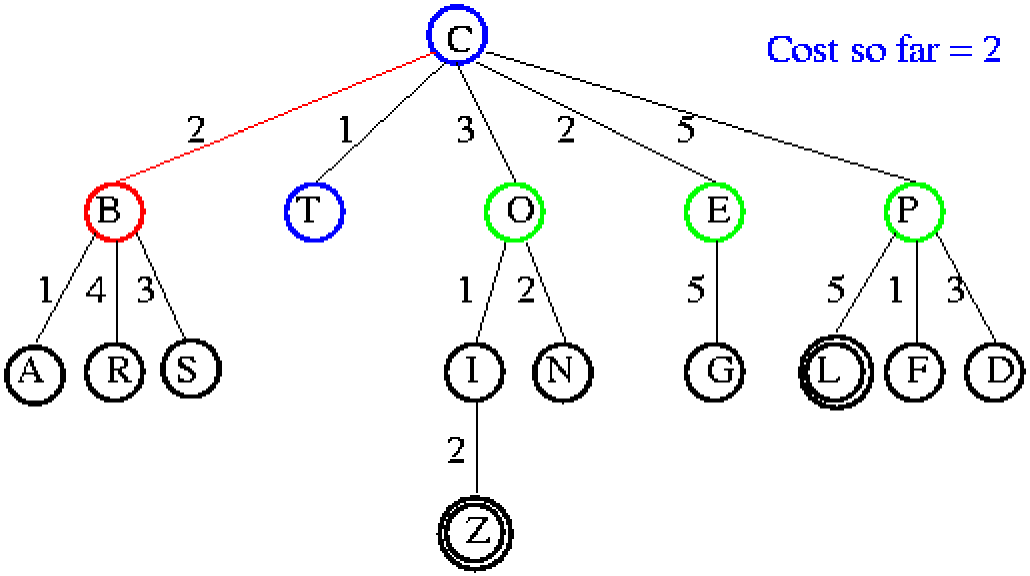
Open list: T(1) B(2) E(2) O(3) P(5)

UCS Example



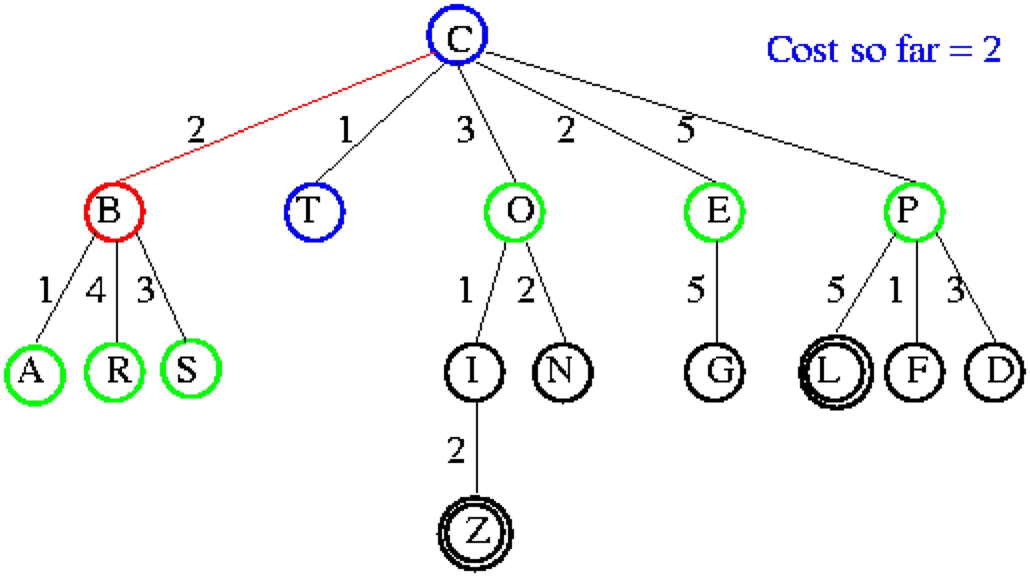
Open list: B(2) E(2) O(3) P(5)

UCS Example



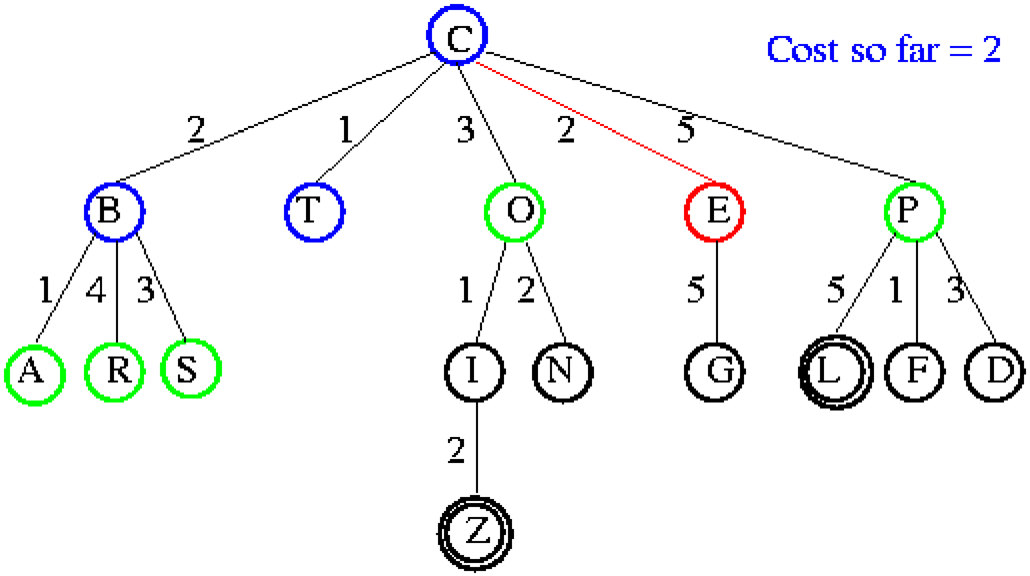
Open list: E(2) O(3) P(5)

UCS Example



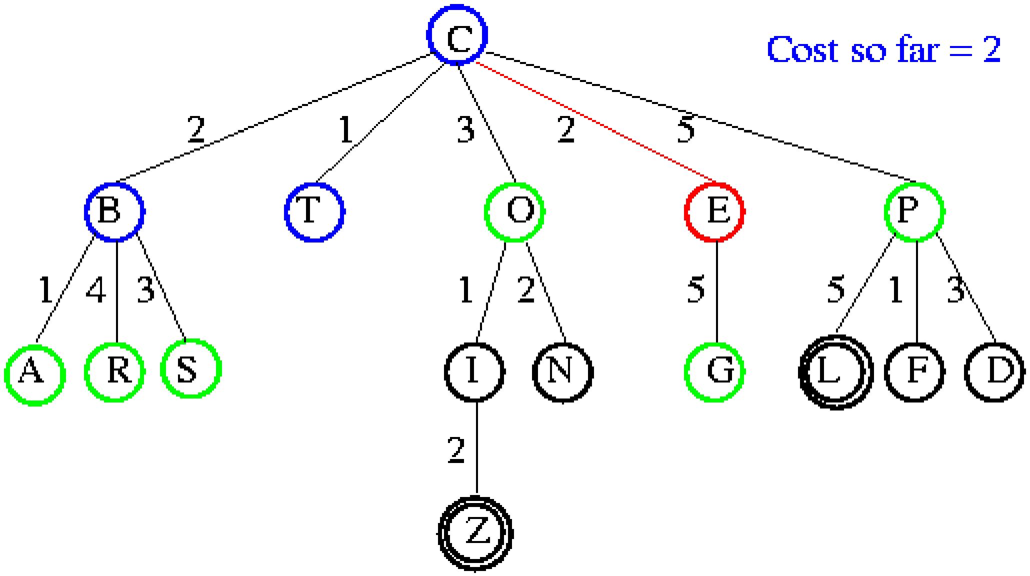
Open list: E(2) O(3) A(3) S(5) P(5) R(6)

UCS Example



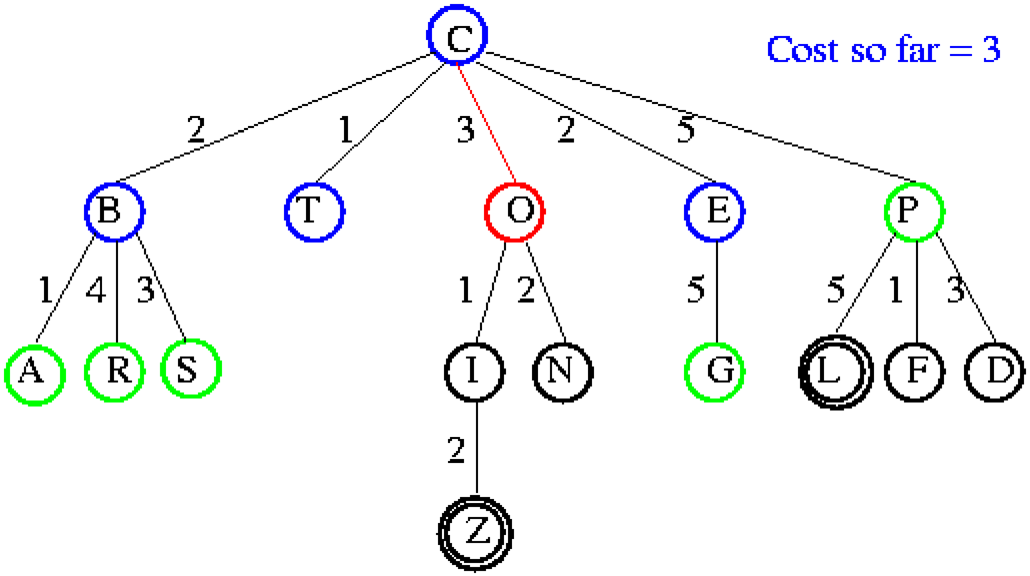
Open list: O(3) A(3) S(5) P(5) R(6)

UCS Example



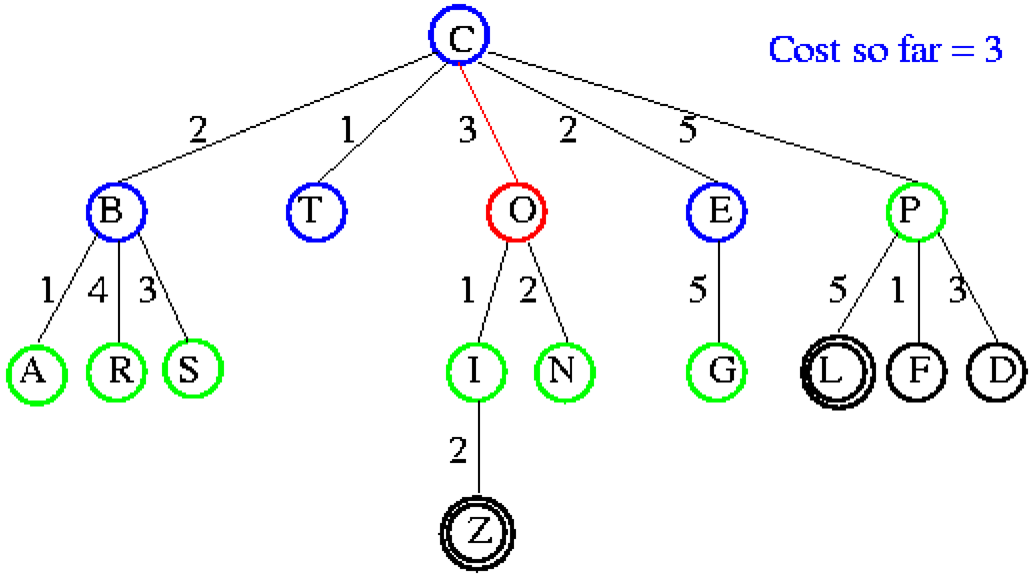
Open list: O(3) A(3) S(5) P(5) R(6) G(10)

UCS Example



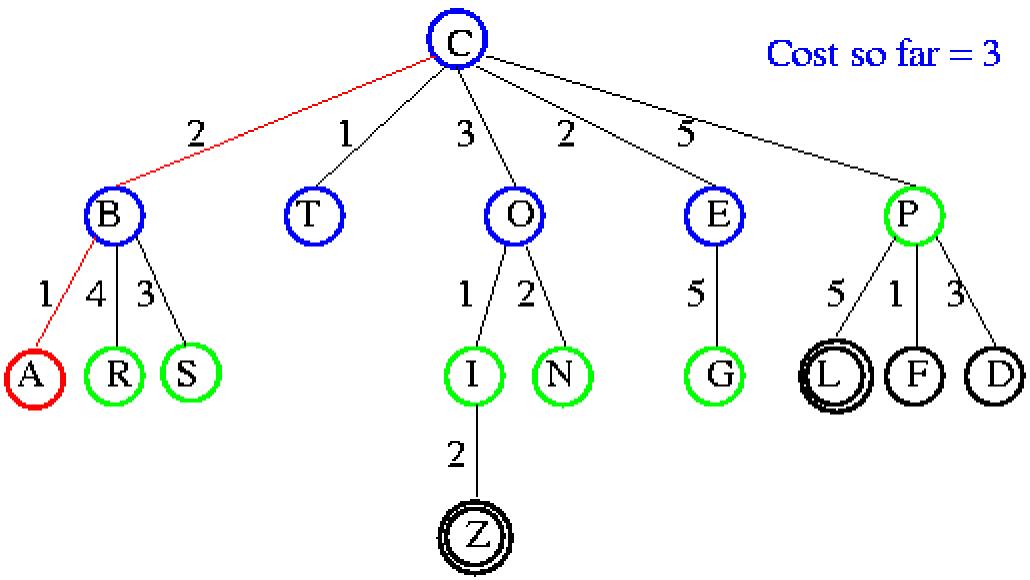
Open list: A(3) S(5) P(5) R(6) G(10)

UCS Example



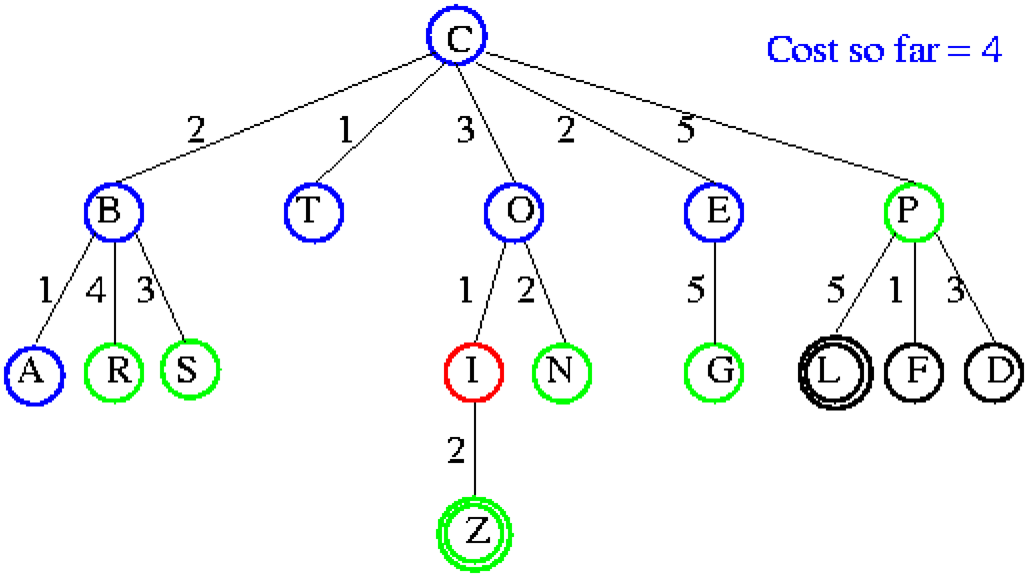
Open list: A(3) I(4) S(5) N(5) P(5) R(6) G(10)

UCS Example



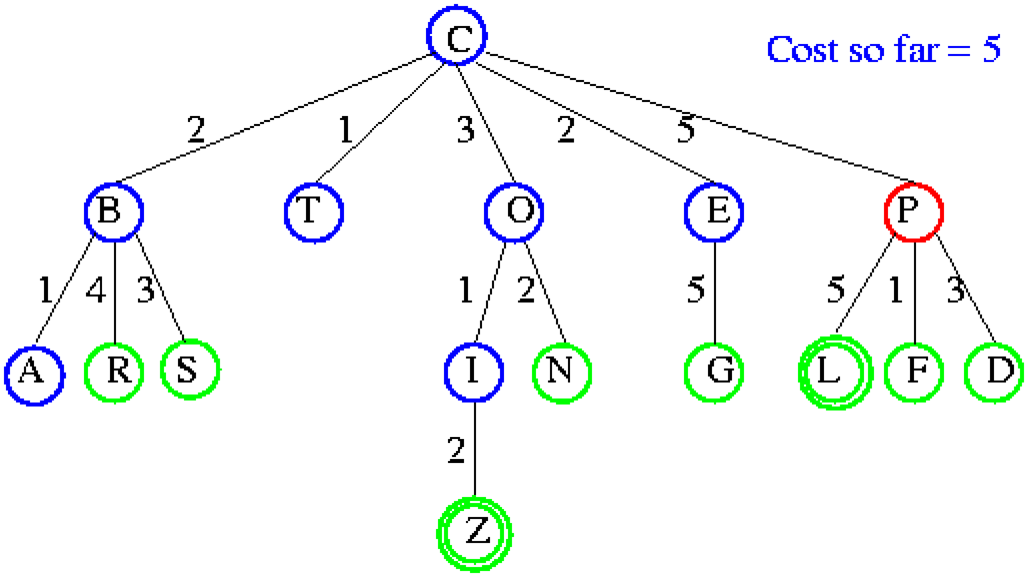
Open list: I(4) P(5) S(5) N(5) R(6) G(10)

UCS Example



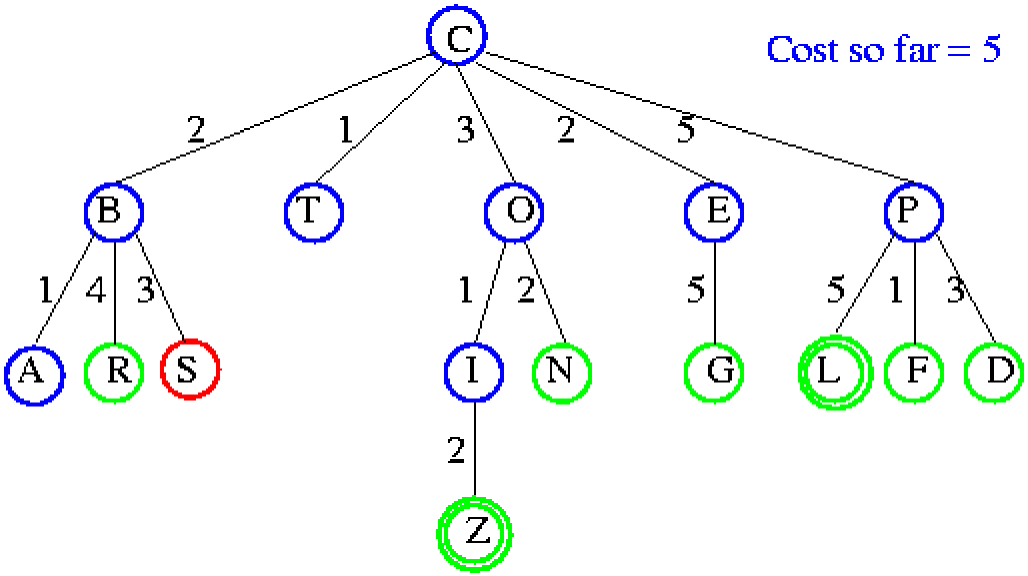
Open list: P(5) S(5) N(5) R(6) Z(6) G(10)

UCS Example



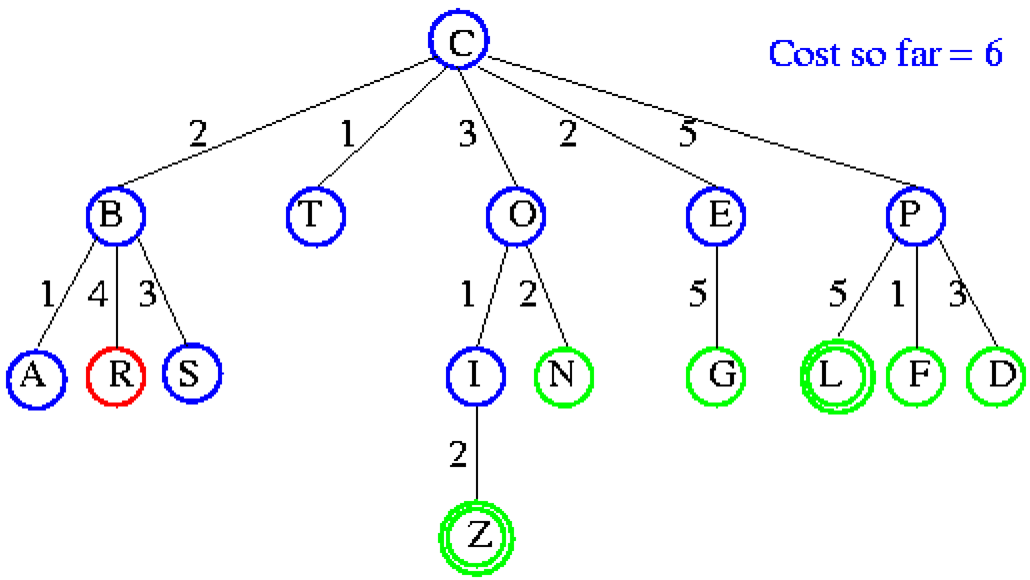
Open list: S(5) N(5) R(6) Z(6) F(6) D(8) G(10) L(10)

UCS Example



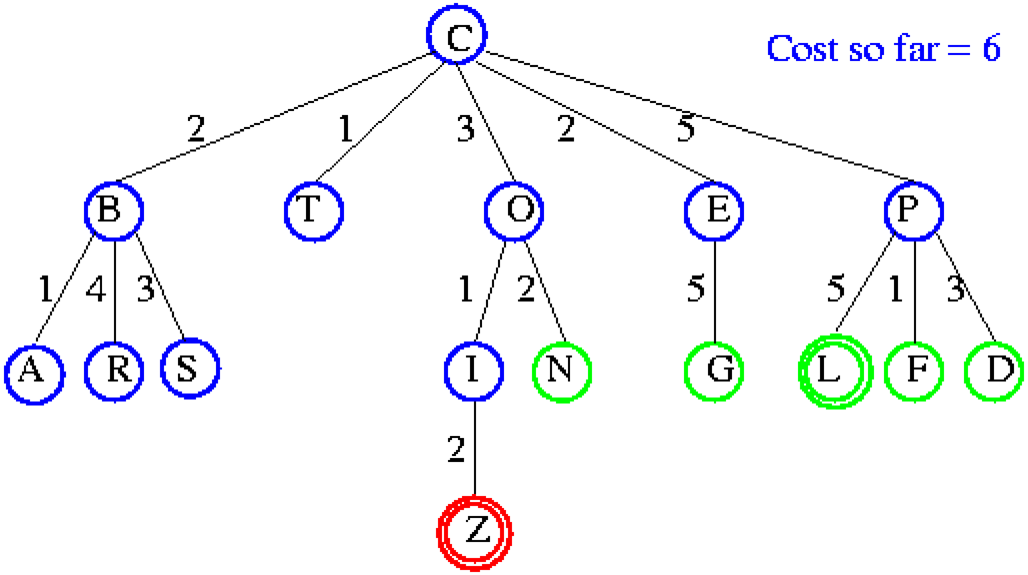
Open list: N(5) R(6) Z(6) F(6) D(8) G(10) L(10)

UCS Example



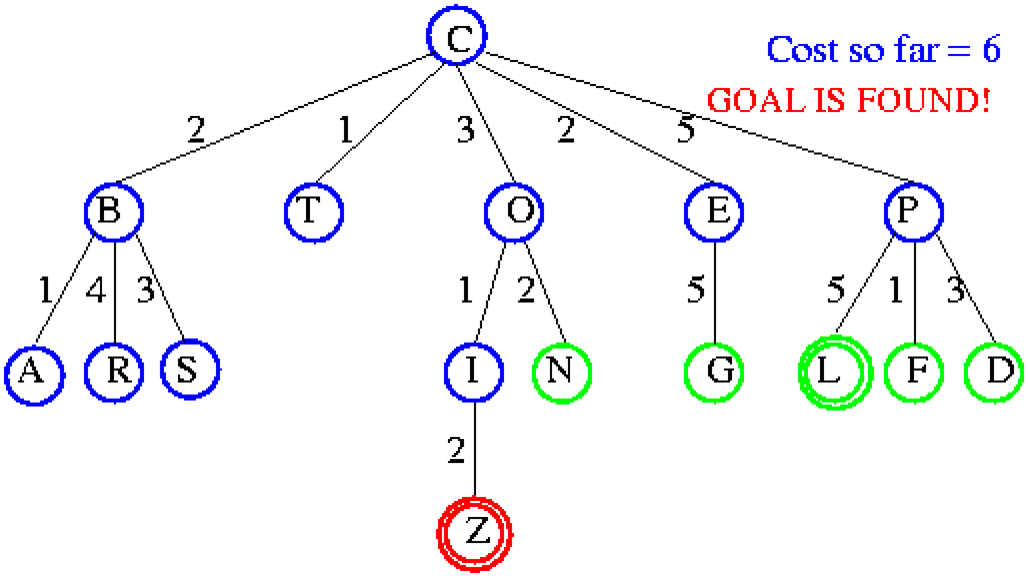
Open list: Z(6) F(6) D(8) G(10) L(10)

UCS Example



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

UCS Example


Cost so far = 6
GOAL IS FOUND!
C
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

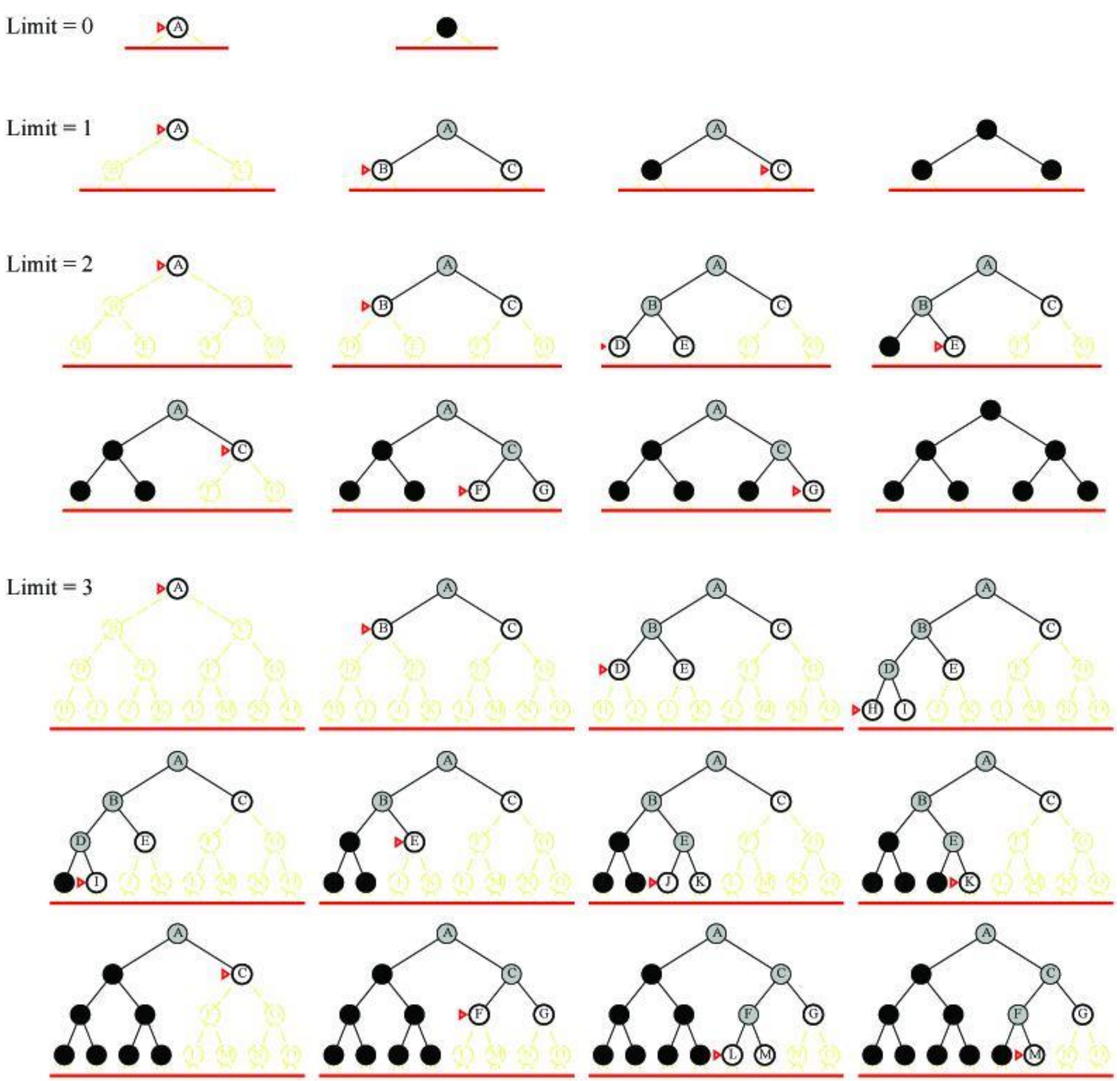
Comparison of Search Techniques

	DFS	BFS	UCS
Complete	N	Y	Y
Optimal	N	N	Y
Heuristic	N	N	N
Time	b^m	b^{d+1}	b^m
Space	bm	b^{d+1}	b^m

IDS
(Iterative Deepening Search)

- DFS with depth bound
- QueuingFn is enqueue at front as with DFS
  - Expand(state) only returns children such that depth(child) <= threshold
  - This prevents search from going down infinite path
- First threshold is 1
  - If do not find solution, increment threshold and repeat

Examples



Analysis

- What about the repeated work?
- Time complexity (number of generated nodes)
  - $[b] + [b + b^2] + .. + [b + b^2 + .. + b^d]$
  - $(d)b + (d-1)\ b^{2\ +} \ldots + (1)\ b^d$
  - $O(b^d)$

Analysis

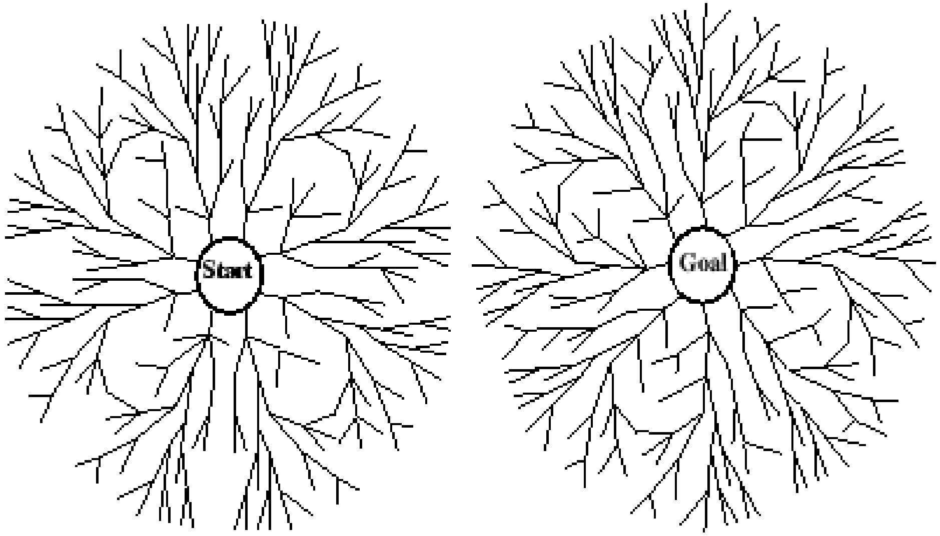
- Repeated work is approximately 1/b of total work
  - Negligible
  - Example: b=10, d=5
  - N(BFS) = 1,111,100
  - N(IDS) = 123,450
- Features
  - Shortest solution, not necessarily least cost
  - Is there a better way to decide threshold? (IDA*)

Comparison of Search Techniques

	DFS	BFS	UCS	IDS
Complete	N	Y	Y	Y
Optimal	N	N	Y	N
Heuristic	N	N	N	N
Time	b^m	b^{d+1}	b^m	b^d
Space	bm	b^{d+1}	b^m	bd

Bidirectional Search

- Search forward from initial state to goal AND backward from goal state to initial state
- Can prune many options
- Considerations
  - Which goal state(s) to use
  - How determine when searches overlap
  - Which search to use for each direction
  - Here, two BFS searches
- Time and space is $O(b^{d/2})$



Uninformed Search Algorithms Comparison

Criterion	Breadth-First	Uniform-cost	Depth-First	Depth-limited	Iterative deepening	Bidirectional search
Complete?	YES*	YES*	NO	YES, if $l \geq d$	YES	YES*
Time	b^{d+1}	$b^{C^*/e}$	b^m	b^l	b^d	$b^{d/2}$
Space	b^{d+1}	$b^{C^*/e}$	bm	bl	bd	$b^{d/2}$
Optimal?	YES*	YES*	NO	NO	YES	YES