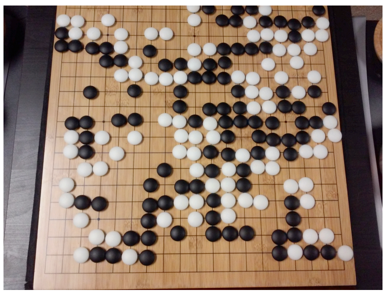
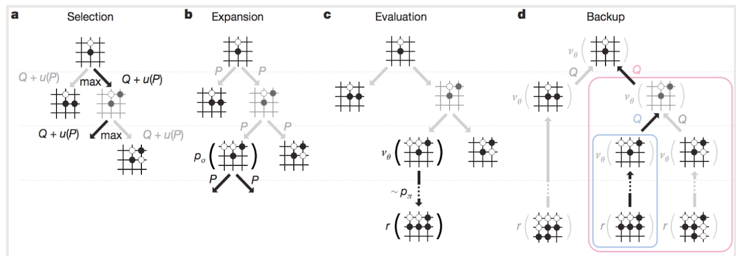
Artificial Intelligence class 3: Search (Ch. 3.1–3.3)





Some material adopted from notes by Charles R. Dyer, University of Wisconsin-Madison

1

Bookkeeping

- Final reminder: readings, HW, etc. are on the class schedule
  - tiny.cc/ai-schedule
- Final reminder: the class Discord is available and useful
  - tiny.cc/ai-discord
- HW1 is out, please verify that you can find it

2

Bits From Last Time

- Sequential: Require memory of past actions to determine next best action
  - Or: current action can influence all future actions
- Episodic: A series of one-shot actions
  - Only the current percept(s) are relevant
  - Sensing/acting in episode(t) is independent of episode(t-1)
- Single- vs. multi-agent: Is "your" agent the only one affecting the world?

} Properties of the agent's *environment* (P**E**AS)

en.wikibooks.org/wiki/Artificial_Intelligence/AI_Agents_and_their_Environments
jeffclune.com/courses/media/courses/2014-Fall-AI/lectures/L04-AI-2014.pdf

3

Some Examples

Agent Type	Performance Measure	Environment	Actuators	Sensors
Robot soccer player	Winning game, goals for/against	Field, ball, own team, other team, own body	Devices (e.g., legs) for locomotion and kicking	Camera, touch sensors, accelerometers, orientation sensors, wheel/joint encoders
Internet book-shopping agent	Obtain requested/ Interesting books, minimize expenditure	Internet	Follow link, enter/submit data in fields, display to user	Web pages, user requests

PEAS

Task Environment	Observable	Deterministic	Episodic	Static	Discrete	Agents
Robot soccer	Partially	Stochastic	Sequential	Dynamic	Continuous	Multi
Internet book-shopping	Partially	Deterministic	Sequential	Static	Discrete	Single

Environment

4

Pre-Reading: Questions?

- Search (a.k.a. state-space search)
- Concepts:
  - Initial state
  - State space graph
  - Goal test (cf. goal)
  - Actions
  - Transition model
  - Step cost
  - Path cost
  - Solution / optimal solution
- Open-loop/closed-loop systems
- Expanding vs. generating a state
- The frontier (a.k.a. open list)

5

What's a "State"?

- The current value of everything in the agent's "world"
  - "State space": all possible states
- Everything in the problem representation
- Values of all **parameters** at a particular point in time
- Examples:
  - Chess board: 8x8 grid, location of all pieces
  - Tic-tac-toe: 3x3 grid, whether each is X, O, or open
  - Robot soccer: Location of all players, location of ball, possibly last known trajectory of all players (if sequential)
  - Travel: Cities, distances between cities, agent's current city

6

Today's Class

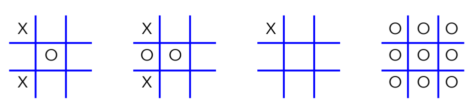
- Representing states and operators
- Example problems
- Generic state-space search algorithm

- Everything in AI comes down to search.
- Goal: understand search, and understand why.

7

Why Search?

- Traditional (non-AI) problems are likely tractable.
  - Either they can be solved by listing all possible states…



…

    - Tic-tac-toe: 3^9 = 19,683 states (3 values for each cell, nine cells)*
    - Small enough that a computer can explore all possible choices during play
  - Or there's a mechanical approach to finding a solution

345,781,000 × 234,567,431,000

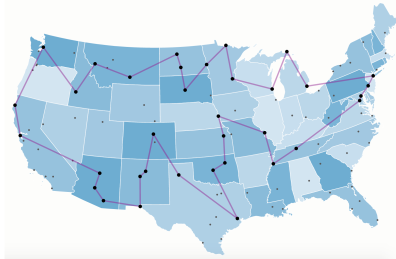
    - Can't memorize the space of answers, but you don't need to

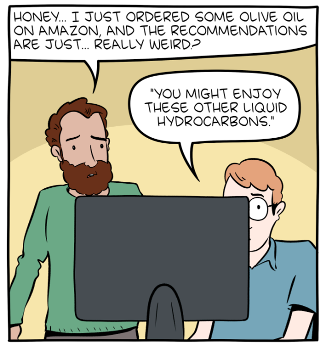
* Of course, there are fewer *valid* states

8

Why Search? (2)

- “Intelligent” problems are usually intractable.
  - Either the state space is too large to enumerate…





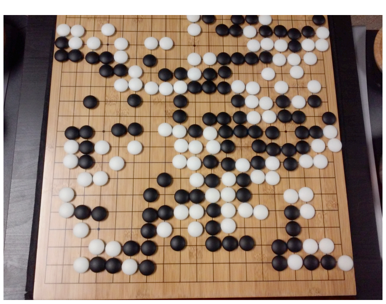
  - We don’t know what a good solution is until we find it…
  - Or, somehow, we have more states than we can explore.

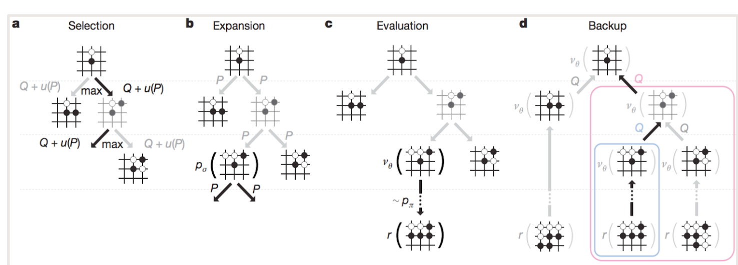
examples.gurobi.com/traveling-salesman-problem, en.wikipedia.org/wiki/Free_Internet_Chess_Server, www.smbc-comics.com/comic/recommendations

9

Why Search? (3)

- We can’t search intractable problems exhaustively, so we must consider them cleverly.
- Understanding the problem space is the first step.



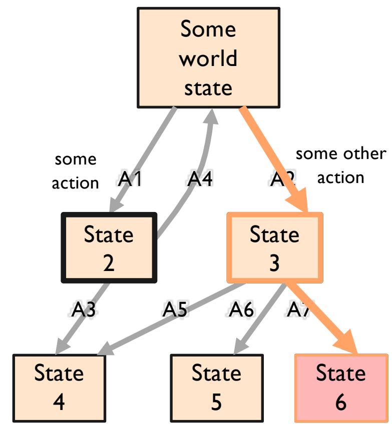


needpix.com, machinelearnings.co/understanding-alphago-948607845bb1

10

Search: The Core Idea

- For any problem:
  - World is (always) in some state
  - Agents take actions, which change the state
- We need a sequence of actions that gets the world into a particular goal state.
- To find it, we search the space of actions and states.

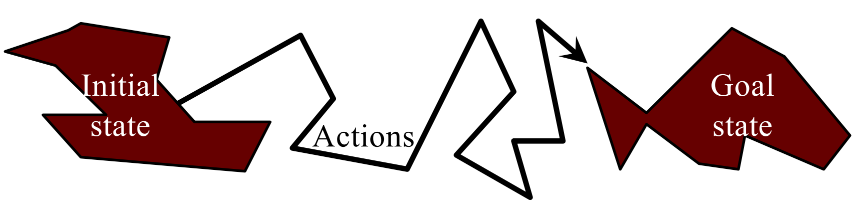


- **Searching** is not (always) the same as **doing**!

11

Building Goal-Based Agents

- To build a goal-based agent we need to decide:
  - What is the goal to be achieved?
  - What are the possible actions?
  - What relevant information must be encoded…
    - To describe the state of the world?
    - To describe the available transitions?
    - To solve the problem?



13

What is the Goal?

- A situation we want to achieve
- A set of properties that we want to hold
- Must define a "**goal test**" (a function over states)
  - What does it mean to achieve it?
  - Have we done so?
- Defining goals is a hard question that is rarely tackled in AI!
  - Often, we assume the system designer or user will specify the goal
- For people, we stress the importance of establishing clear goals as the first step towards solving a problem.
  - What are your goals?
  - What problem(s) are you trying to solve?

14

What Are Actions?

- Primitive actions or events:
  - Make changes in the world
  - In order to achieve a (sub)goal
  - Actions are also known as operators or moves
- Examples:

Low-level:
- Chess: "advance a pawn"
- Navigation: "take a step"
- Finance: "sell 10% of stock X"

High-level :
- Chess: "clear a path for a queen"
- Navigation: "go home"
- Finance: "sell best-return shares"

15

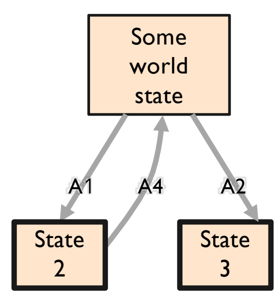
Actions and Determinism

- In a deterministic world there is no uncertainty in an action's effects
- Current world state + chosen action fully specifies:
- Whether that action can be done in current world
  - Is it applicable? (E.g.: Do I own any of stock X to sell?)
  - Is it legal? (E.g.: Can't just move a pawn sideways.)
- World state after action is performed

16

Representing Actions

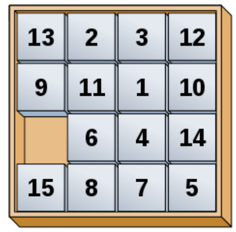
- Actions here are:
  - Discrete events
  - That occur at an instant of time
- For example:
  - State: "Mary is in class"
  - Action "Go home"
  - New state: "Mary is at home"
  - There is no representation of a state where she is in between (i.e., in the state of "going home").

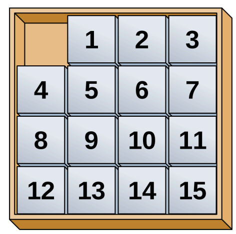


17

Sliding Tile Puzzles

- 15-puzzles, 8-puzzles
- How do we represent states?
- How do we represent actions?
  - Tile-1 moves north
  - Tile-1 moves west
  - Tile-1 moves east
  - Tile-1 moves south
  - Tile-2 moves north
  - Tile-2 moves west
  - …





commons.wikimedia.org/wiki/File:15-puzzle-shuffled.svg, commons.wikimedia.org/wiki/File:15-puzzle-loyd-bis2.svg

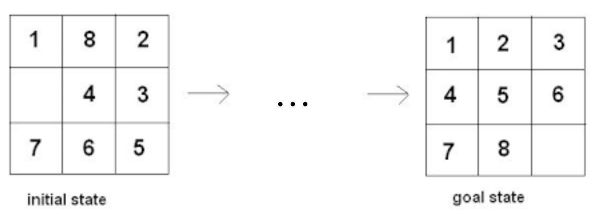
18

Representing Actions

- Number of actions / operators depends on representation used in describing a state
- 8-puzzle:
  - Could specify 4 possible moves (actions) for each of the 8 tiles:

    4*8=32 operators.
  - Or, could specify four moves for the "blank" square:

    4 operators!
- Careful representation can simplify a problem!



19

Representing States

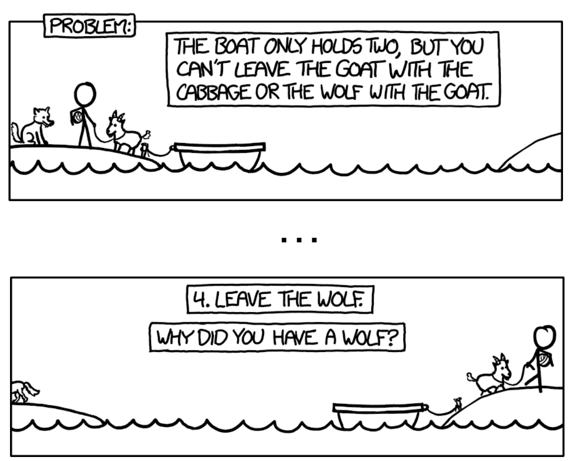
- What information about the world sufficiently describes all aspects relevant to solving the goal?
- That is: what knowledge must be in a state description to adequately describe the current state of the world?
- The size of a problem is usually described in terms of the number of states that are possible
  - Tic-Tac-Toe has about 3^9 states.
  - Checkers has about 10^{40} states.
  - Rubik's Cube has about 10^{19} states.
  - Chess has about 10^{120} states in a typical game.

This is *ten quintillion* states.

20

Some Example Problems

- Toy problems and micro-worlds
  - 8-Puzzle
  - Boat Problems
  - Cryptarithmetic
  - Remove 5 Sticks
  - Water Jug Problem

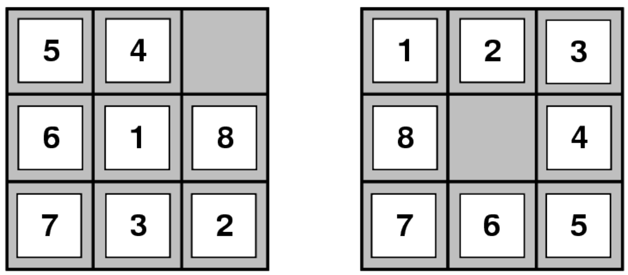


https://xkcd.com/1134

22

8-Puzzle

- Given an initial configuration of 8 sliding numbered tiles on a 3 x 3 board, move the tiles in such a way so as to produce a desired goal configuration of the tiles.

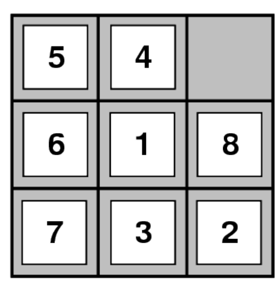


Start State   Goal State

23

8-Puzzle

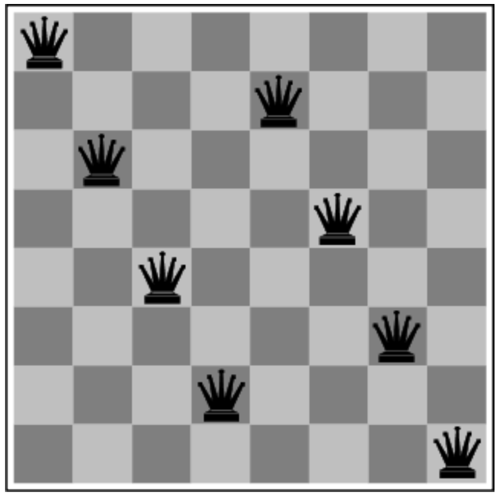
- **State:** 3 x 3 array describing where tiles are
- **Operators:** Move blank square Left, Right, Up or Down
  - This is a more efficient encoding of the operators!
- **Initial State:** Starting configuration of the board
- **Goal:** Some specific board configuration



24

The 8-Queens Problem

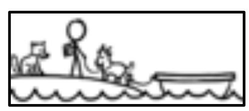
- Place eight (or N) queens on a chessboard such that no queen can reach any other



25

Boat Problems

- 1 sheep, 1 wolf, 1 cabbage, 1 boat
- Goal: Move everything across the river.
- Constraints:
  - The boat can hold you plus one thing.
  - Wolf can never be alone with sheep.
  - Sheep can never be alone with cabbage.
- State: location of sheep, wolf, cabbage on shores and boat.
- Operators: Move ferry containing some set of occupants across the river (in either direction) to the other side.

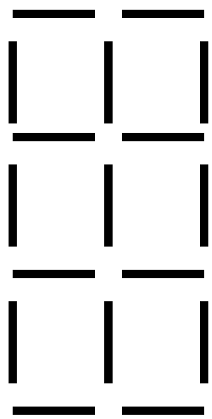


https://xkcd.com/1134

26

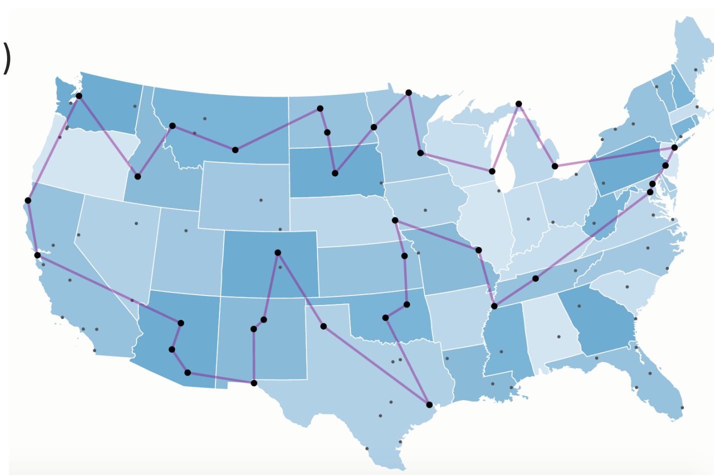
Remove 5 Sticks

- Given the following configuration of sticks, remove exactly 5 sticks in such a way that the remaining configuration forms exactly 3 squares.



Some Real-World Problems

- Route finding
- Touring (traveling salesman)
- Logistics
- VLSI layout
- Robot navigation
- Learning



Knowledge Representation Issues

- What's in a state?
  - Is the color of the tiles relevant to solving an 8-puzzle?
  - Is sunspot activity relevant to predicting the stock market?
- What to represent is a very hard problem!
  - Usually left to the system designer to specify.
- What level of abstraction to describe the world?
  - Too fine-grained and we "miss the forest for the trees"
  - Too coarse-grained and we miss critical information

30

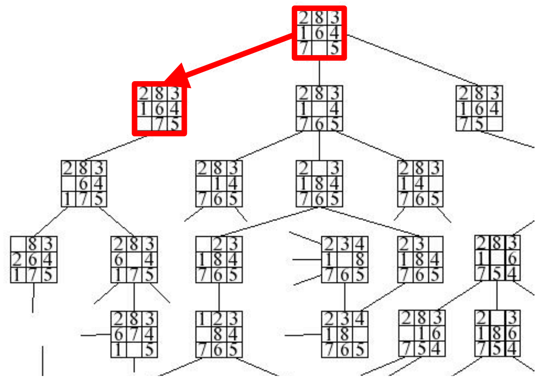
Knowledge Representation Issues

- Number of states depends on:
  - Representation choices
  - Level of abstraction
- In the Remove-5-Sticks problem:
  - If we represent individual sticks, then there are 17-choose-5 possible ways of removing 5 sticks (6188)
  - If we represent the "squares" defined by 4 sticks, there are 6 squares initially and we must remove 3
  - So, 6-choose-3 ways of removing 3 squares (20)

31

Formalizing Search in a State Space

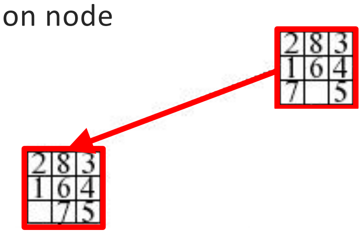
- A state space is a graph (V, E):
  - V is a set of nodes (states)
  - E is a set of arcs (actions)
  - Each arc is directed from a node to another node
- How does that work for 8-puzzle?



32

Formalizing Search in a State Space

- V: A node is a data structure that contains:
  - State description
  - Bookkeeping information: parent(s) of the node, name of operator that generated the node from that parent, etc.
- E: Each arc is an instance (single occurrence) of one operator.
  - When operator is applied to the arc's source node (state), then
  - Resulting state is associated with the arc's destination node



33

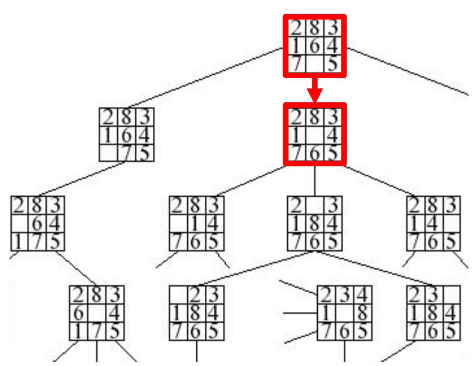
Formalizing Search

- Each arc has a fixed, positive cost
  - Corresponding to the cost of the operator
  - What is “cost” of doing that action?
- Each node has a set of **successor nodes**
  - Corresponding to all operators (actions) that can apply at source node’s state
  - **Expanding** a node is generating successor nodes, and adding them (and associated arcs) to the state-space graph
  - We don’t know all states initially – we have to apply operators and calculate the successor nodes

34

Formalizing Search II

- One or more nodes are designated as start nodes
- A goal test predicate is applied to a state to determine if its associated node is a goal node



35

Water Jug Problem as Search

Given a full 5-gallon jug and an empty 2-gallon jug, the goal is to fill the 2-gallon jug with exactly one gallon of water.

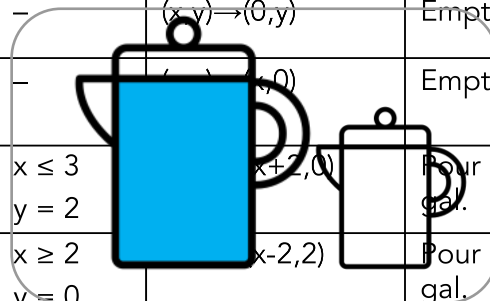
State = (x,y), where x is the number of gallons of water in the 5-gallon jug and y is # of gallons in the 2-gallon jug

Initial State = (5,0)

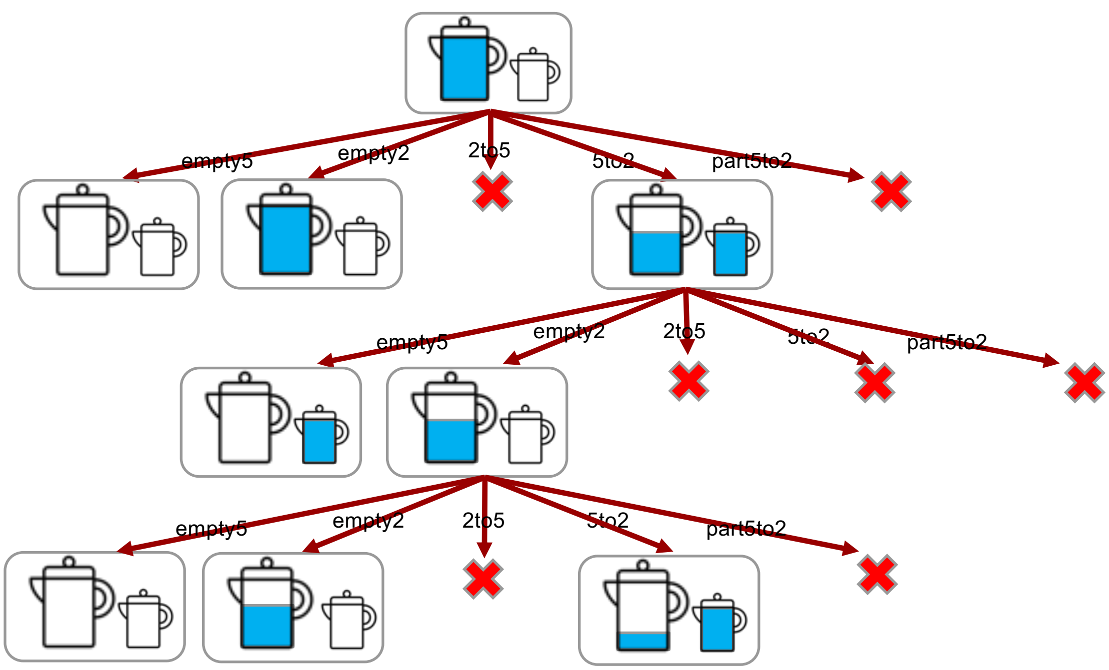
Goal State = (*,1)
(* means any amount)

Operator table

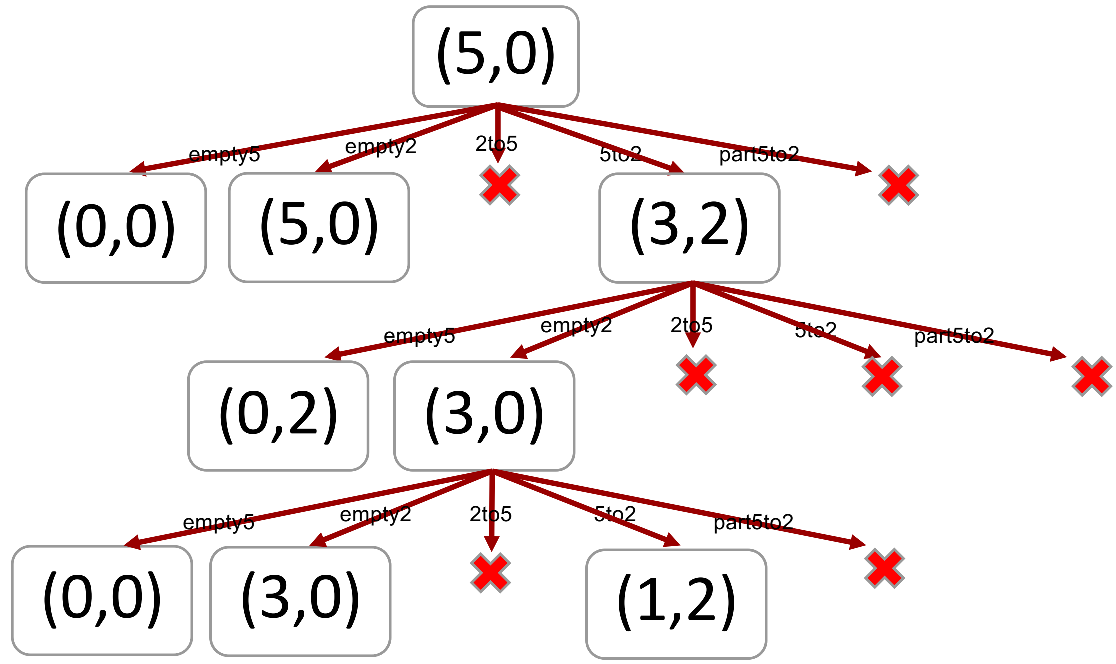
Name	Cond.	Transition	Effect
Empty5	–	(x,y)→(0,y)	Empty 5-gal. jug
Empty2	–	(x,y)→(x,0)	Empty 2-gal. jug
2to5	x ≤ 3 y = 2	(x,y)→(x+2,0)	Pour 2-gal. into 5-gal.
5to2	x ≥ 2 y = 0	(x,y)→(x-2,2)	Pour 5-gal. into 2-gal.
5to2part	y < 2 x = 1	(1,y)→(0,y+1)	Pour partial 5-gal. into 2-gal.



36



37



38

Formalizing Search III

- A **solution** is a sequence of operators that is associated with a path in a state space from a start node to a goal node.
  - 5to2, empty2, 5to2, empty2, 5to2part
- The **cost** of a solution is the sum of the arc costs on the solution path.
  - If all arcs have the same (unit) cost, then the solution cost is just the length of the solution (number of steps / state transitions)

39

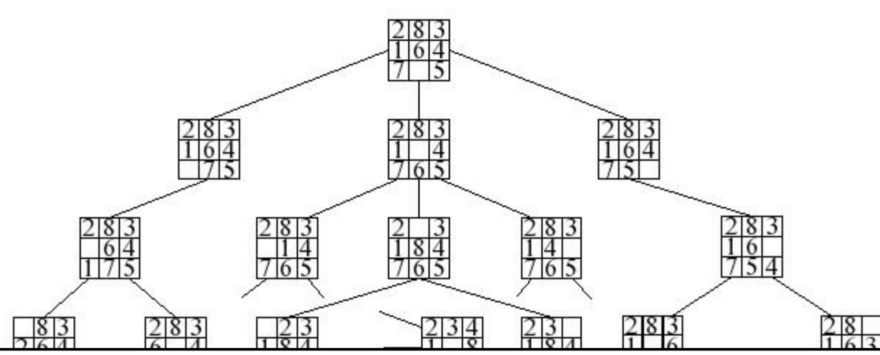
Formalizing Search IV

- State-space search: searching through a state space for a solution by making explicit a sufficient portion of an implicit state-space graph to find a goal node
  - Initially V={S}, where S is the start node
  - When S is expanded, its successors are generated; those nodes are added to V and the arcs are added to E
  - This process continues until a goal node is found
- It isn't usually practical to represent entire space

40

Formalizing Search V

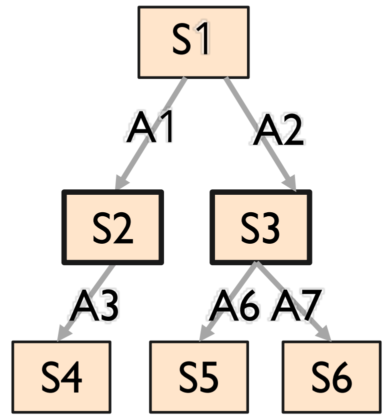
- Each node implicitly or explicitly represents a partial solution path (and its cost) from start node to given node.
  - In general, from a node there are many possible paths (and therefore solutions) that have this partial path as a prefix



41

State-Space Search Algorithm

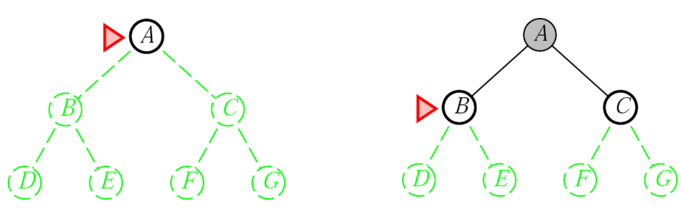
```
function general-search (problem, QUEUEING-FUNCTION)
    ;; problem describes start state, operators, goal test, and operator costs
    ;; queueing-function is a comparator function that ranks two states
    ;; returns either a goal node or failure

nodes = MAKE-QUEUE(MAKE-NODE(problem.INITIAL-STATE))

loop
      if EMPTY(nodes) then return "failure"
      node = REMOVE-FRONT(nodes)
      if problem.GOAL-TEST(node.STATE) succeeds
        then return node
      nodes = QUEUEING-FUNCTION(nodes, EXPAND(node,
            problem.OPERATORS))
end
      ;; Note: The goal test is NOT done when nodes are generated
      ;; Note: This algorithm does not detect loops
```



42

Generation vs. Expansion

- **Selecting** a state means making that node current
- **Expanding** the current state means applying every legal action to the current state
  - Which generates a new set of nodes

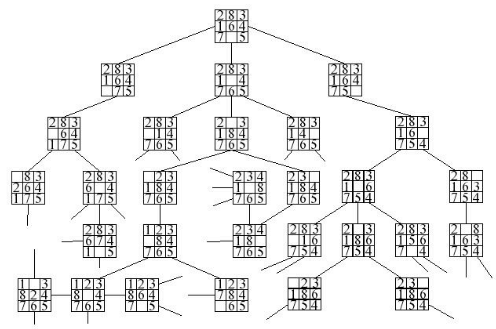


R&N pg. 68, 80

43

Key Procedures

- EXPAND
  - Generate all successor nodes of a given node
    - “What nodes can I reach from here (by taking what actions)?”
- GOAL-TEST
  - Test if state satisfies goal conditions
- QUEUEING-FUNCTION
  - Used to maintain a ranked list of nodes that are candidates for expansion
    - “What should I explore next?”



44

Algorithm Bookkeeping

- Typical node data structure includes:
  - State at this node
  - Parent node
  - Operator applied to get to this node
  - Depth of this node
    - That is, number of operator applications since initial state
  - Cost of the path
    - Sum of each operator application so far

45

Some Issues

- Search process constructs a search tree, where:
  - Root is the initial state and
  - Leaf nodes are nodes that are either:
    - Not yet expanded (i.e., they are in the list “nodes”) or
    - Have no successors (i.e., they're “dead ends”, because no operators can be applied, but they are not goals)
- Search tree may be infinite
  - Even for small search space
  - How?

46

Some Issues

- Return a path or a node depending on problem
  - In 8-queens return a node
  - 8-puzzle return a path
  - What about Sheep & Wolves?
- Changing definition of Queueing-Function → different search strategies
  - How do you choose what to expand next?

47

Evaluating Search Strategies

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

48

Summary

- Search is at the heart of AI.
- Formalizing states, actions, &c. makes them searchable.

49

Class Exercise

- Representing a Sudoku puzzle as a search space
  - What are the states?
  - What are the operators?
  - What are the constraints (on operator application)?
  - What is the description of the goal state?
- Let's try it!

	3		
			1
3			
		2	

50

Sudoku, Naïvely

- **State space:** 4x4 matrix, divided into four 2x2 matrices: A, B, C, D, cells containing values [1-4]
- **Operators:**
  - Put a **2** in square <x,y>
  - **Preconditions:**
    - <x,y> is empty 1
    - <x, (y±1)> ≠ 2; <x, (y±2)> ≠ 2; … 3 × 4
    - <(x±1), y> ≠ 2; … <(x±3), y> ≠ 2 3
    - **if** <x,y> **in** A, **then** 3 ∉ A; … 4

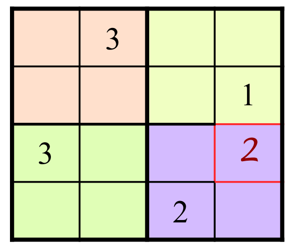
	3		
			1
3			
		2	

- How many operators is that? How many preconditions?
- **Goal:** all blocks are filled

51

Sudoku, Naïvely

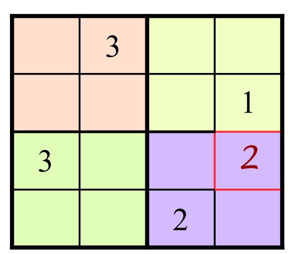
- **State space:** 4x4 matrix, divided into four 2x2 matrices: A, B, C, D, cells containing values [1-4]
- **Operators:**
  - Put a **2** in square <x,y>
  - Preconditions:
    - <x,y> is empty — 1
    - <x, (y±1)> ≠ 2; <x, (y±2)> ≠ 2; … — 3 × 4
    - <(x±1), y> ≠ 2; … <(x±3), y> ≠ 2 — 3
    - **if** <x,y> **in** A, **then** 3 ∉ A; … — 4
- How many operators is that?
- **Goal:** all blocks are filled



52

Sudoku, Naïvely

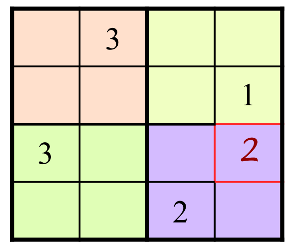
- **State space:** 4x4 matrix, divided into four 2x2 matrices: A, B, C, D, cells containing values [1-4]
- **Operators:**
  - Put a **2** in square <x,y>
  - Preconditions:
    - ✓ **<x,y> is empty** — 1
    - <x, (y±1)> ≠ 2; <x, (y±2)> ≠ 2; … — 3 × 4
    - <(x±1), y> ≠ 2; … <(x±3), y> ≠ 2 — 3
    - **if** <x,y> **in** A, **then** 3 ∉ A; … — 4
- How many operators is that?
- **Goal:** all blocks are filled



53

Sudoku, Naïvely

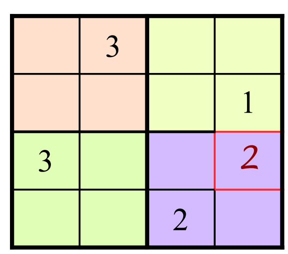
- **State space:** 4x4 matrix, divided into four 2x2 matrices: A, B, C, D, cells containing values [1-4]
- **Operators:**
  - Put a **2** in square <x,y>
  - Preconditions:
    - ✓ <x,y> is empty 1
    - ✓ **<x, (y±1)> ≠ 2; <x, (y±2)> ≠ 2; …** 3 x 4
    - <(x±1), y> ≠ 2; … <(x±3), y> ≠ 2 3
    - **if** <x,y> **in** A, **then** 3 ∉ A; … 4
- How many operators is that?
- **Goal:** all blocks are filled



54

Sudoku, Naïvely

- **State space:** 4x4 matrix, divided into four 2x2 matrices: A, B, C, D, cells containing values [1-4]
- **Operators:**
  - Put a **2** in square <x,y>
  - Preconditions:
    - ✓ <x,y> is empty 1
    - ✓ <x, (y±1)> ≠ 2; <x, (y±2)> ≠ 2; … 3 x 4
    - ✓ **<(x±1), y> ≠ 2; … <(x±3), y> ≠ 2** 3
    - **if** <x,y> **in** A, **then** 3 ∉ A; … 4
- How many operators is that?
- **Goal:** all blocks are filled

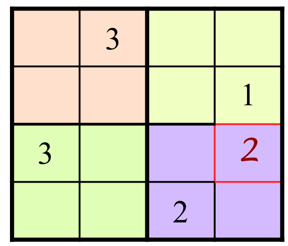


55

Sudoku, Naïvely

- **State space:** 4x4 matrix, divided into four 2x2 matrices: A, B, C, D, cells containing values [1-4]
- **Operators:**
  - Put a **2** in square <x,y>
  - Preconditions:
    - ✓ <x,y> is empty — 1
    - ✓ <x, (y±1)> ≠ 2; <x, (y±2)> ≠ 2; … — 3
    - ✓ <(x±1), y> ≠ 2; … <(x±3), y> ≠ 2 — 3
    - ✗ **if <x,y> in A, then 3 ∉ A; …** — 4

    × 4
- How many operators is that?
- **Goal:** all blocks are filled

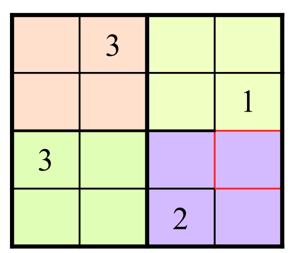


56

Sudoku, Naïvely

- **State space:** 4x4 matrix, divided into four 2x2 matrices: A, B, C, D, cells containing values [1-4]
- **Operators:**
  - Put a **2** in square <x,y>
  - Preconditions:
    - ✓ <x,y> is empty — 1
    - ✓ <x, (y±1)> ≠ 2; <x, (y±2)> ≠ 2; … — 3
    - ✓ <(x±1), y> ≠ 2; … <(x±4), y> ≠ 2 — 3
    - ✗ **if <x,y> in A, then 3 ∉ A; …** — 4

    × 4
- How many operators is that?
- **Goal:** all blocks are filled

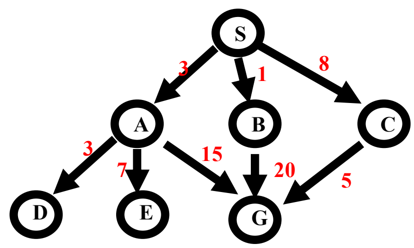


57

Artificial Intelligence

Uninformed Search (Ch. 3.4)

(and a little more formalization)



Some material adapted from slides by Gang Hua of Stevens Institute of Technology
Some material adapted from slides by Charles R. Dyer, University of Wisconsin-Madison

58

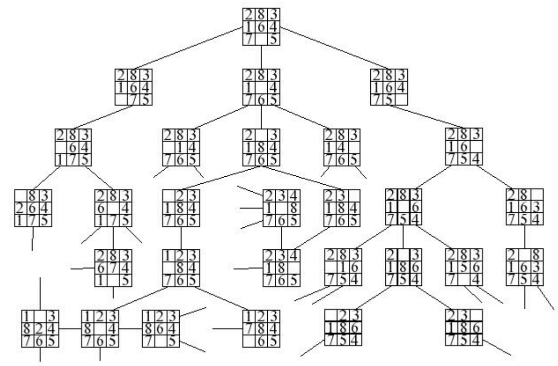
Questions?

- Bread-first, depth-first, uniform cost search
- Generation and expansion
- Goal tests
- Queueing function
- Complexity, completeness, and optimality
- Heuristic functions (for informed search)
- Admissibility

59

Key Procedures

- EXPAND
  - Generate **all** successor nodes of a given node
    - “What nodes can I reach from here (by taking what actions)?”



- GOAL-TEST
  - Test if state satisfies goal conditions
- QUEUEING-FUNCTION
  - Maintain a **ranked** list of nodes that are expansion candidates
    - “What should I explore next?”

60

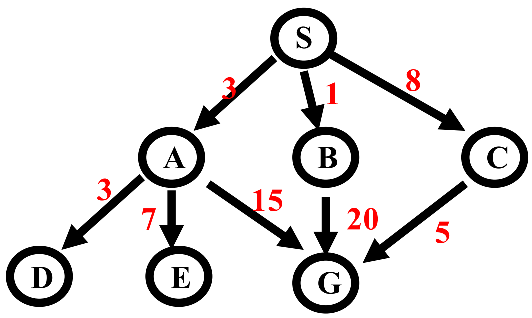
Uninformed vs. Informed Search

- Uninformed (aka “blind”) search
  - Use no information about the “direction” of the goal node(s)
    - No way tell know if we’re “doing well so far”
  - Breadth-first, depth-first, depth-limited, uniform-cost, depth-first iterative deepening, bidirectional
- Informed (aka “heuristic”) search (next class)
  - Use domain information to (try to) (usually) head in the general direction of the goal node(s)
  - Hill climbing, best-first, greedy search, beam search, A, A*

62

Why Apply Goal Test Late?

- Why does it matter when the goal test is applied (expansion time vs. generation time)?
- Optimality and complexity of the algorithms are strongly affected!



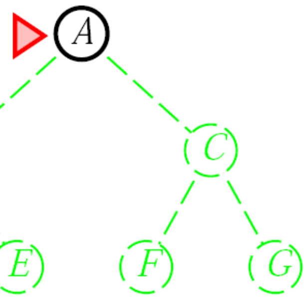
63

Breadth-First

- Enqueue nodes in **FIFO** (first-in, first-out) order
- Characteristics:
  - **Complete** (meaning?)
  - **Optimal** (i.e., admissible) if all operators have the same cost
  - Otherwise, not optimal but finds solution with shortest path length
  - **Exponential time and space complexity**, $O(b^d)$, where:
    - d is the depth of the solution
    - b is the branching factor (number of children) at each node
- Takes a long time to find long-path solutions

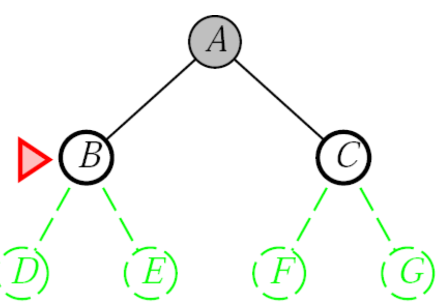
64

BFS



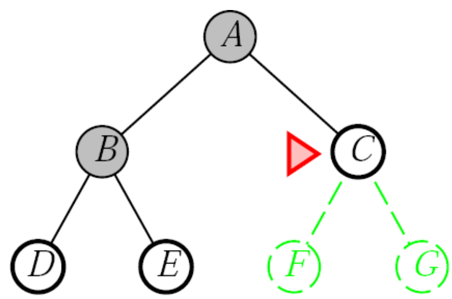
65

BFS



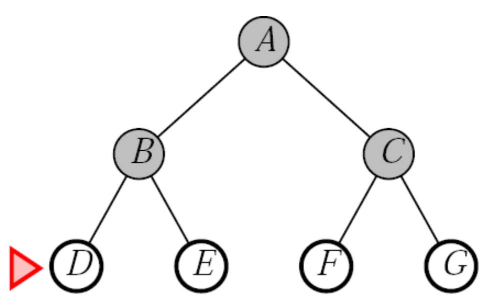
66

BFS



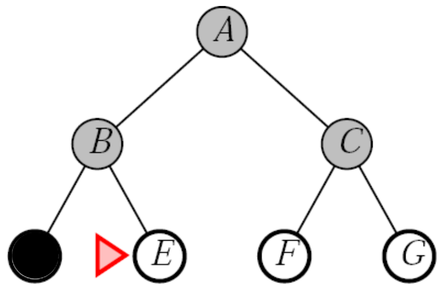
67

BFS



68

BFS



69

Breadth-First: Analysis

- Takes a long time to find long-path solutions
  - Must look at all shorter length possibilities first
  - A complete search tree of depth d where each non-leaf node has b children:
- $1 + b + b^2 + \ldots + b^d = (b^{d+1} - 1)/(b-1)$ nodes
- **Checks a lot of short-path solutions quickly**

70

Breadth-First: O(Example)

- $1 + b + b^2 + ... + b^d = (b^{d+1} - 1)/(b-1)$ nodes
- Tree where: d=12
- Every node at depths 0, ..., 11 has 10 children (b=10)
- Every node at depth 12 has 0 children
- $1 + 10 + 100 + 1000 + ... + 10^{12} = (10^{13-1})/9 = O(10^{12})$ nodes in the complete search tree
- If BFS expands 1000 nodes/sec and each node uses 100 bytes of storage
  - Will take 35 years to run in the worst case
  - Will use 111 terabytes of memory

71

Depth-First (DFS)

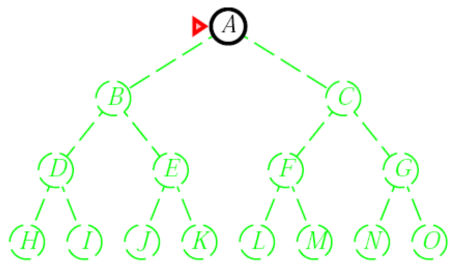
- Enqueue nodes in **LIFO** (last-in, first-out) order
  - That is, nodes used as a stack data structure to order nodes
- Characteristics:
  - **Might not terminate** without a "depth bound"
    - I.e., cutting off search below a fixed depth D ("depth-limited search")
  - **Not complete**
    - With or without cycle detection, and with or without a cutoff depth
  - **Exponential time**, $O(b^d)$, but only **linear space**, O(bd)

Loops?

Infinite search spaces?

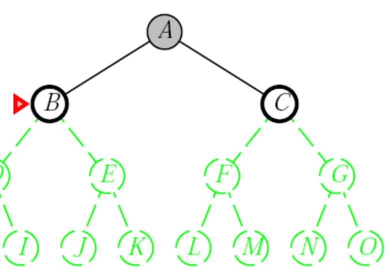
72

DFS



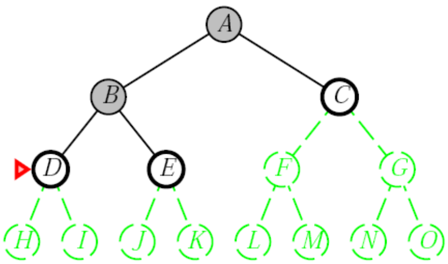
73

DFS



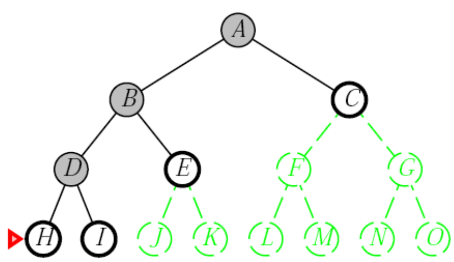
74

DFS



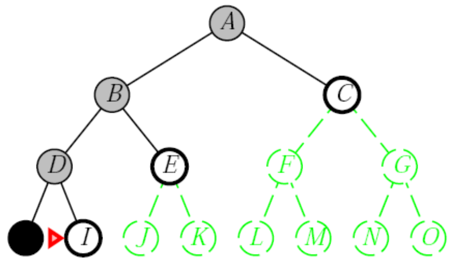
75

DFS



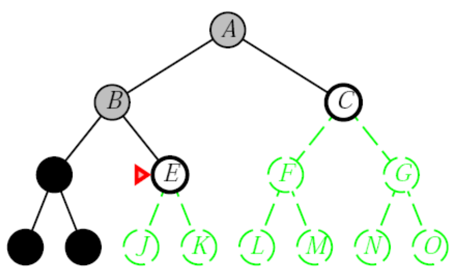
76

DFS



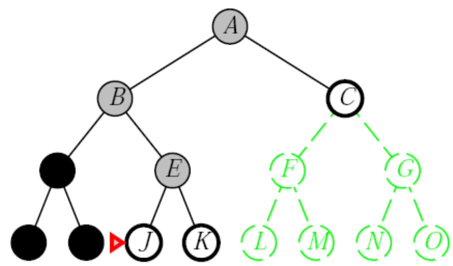
77

DFS



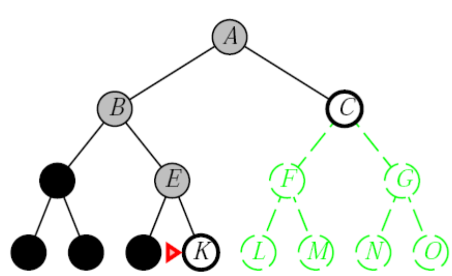
78

DFS



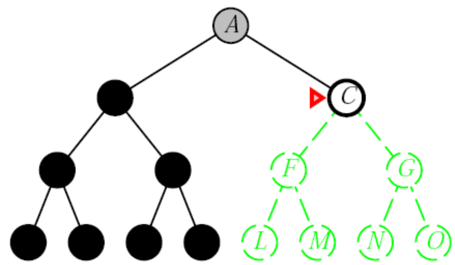
79

DFS



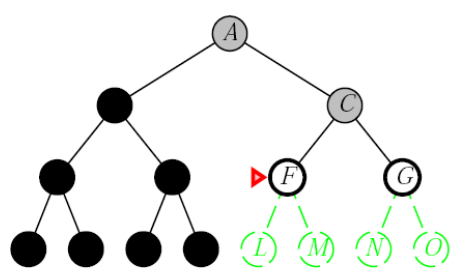
80

DFS



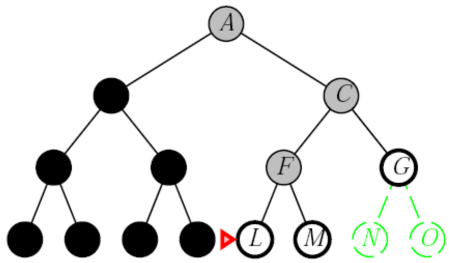
81

DFS



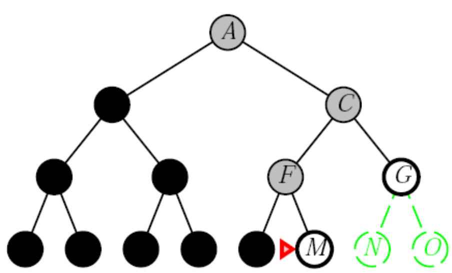
82

DFS



83

DFS



84

Depth-First (DFS): Analysis

- DFS:
  - Can find **long solutions quickly** if lucky
  - And **short solutions slowly** if unlucky
- When search hits a dead end
  - Can only back up one level at a time*
  - Even if the "problem" occurs because of a bad operator choice near the top of the tree
  - Hence, only does "chronological backtracking"
- * Why?

85

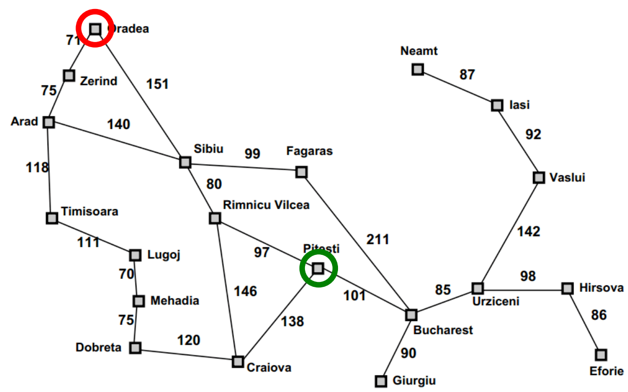
Uniform-Cost (UCS)

- Enqueue nodes by **path cost**:
  - Let g(n) = cost of path from start node to current node *n*
  - Sort nodes by increasing value of *g*
  - Identical to breadth-first search **if** all operators have equal cost
- *"Dijkstra's Algorithm"* in algorithms literature
- *"Branch and Bound Algorithm"* in operations research literature
- **Complete** (*)
- **Optimal/Admissible** (*)
  - Admissibility depends on the goal test being applied *when a node is removed from the nodes list*, not when its parent node is expanded and the node is first generated
- **Exponential time and space complexity**, $O(b^d)$

86

Example: Path Costs

Romania with step costs in km



Straight−line distance to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

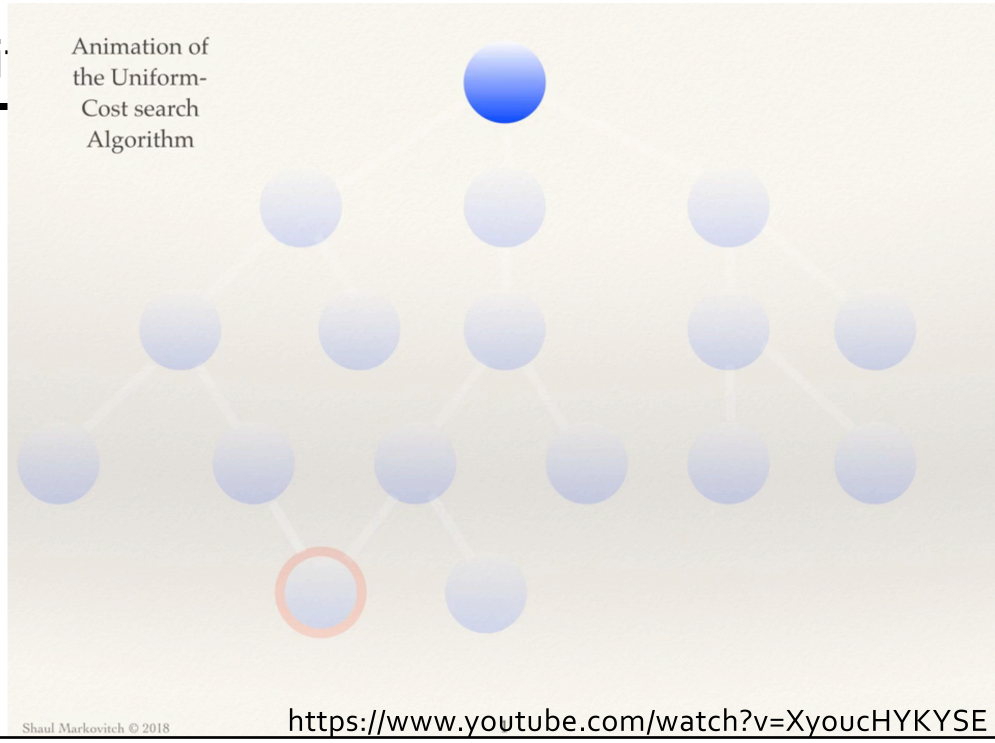
87

UCS Implementation

- For each frontier node, save the total cost of the path from the initial state to that node
- Expand the frontier node with the lowest path cost
- Equivalent to breadth-first if step costs all equal
- Equivalent to Dijkstra’s algorithm in general

88

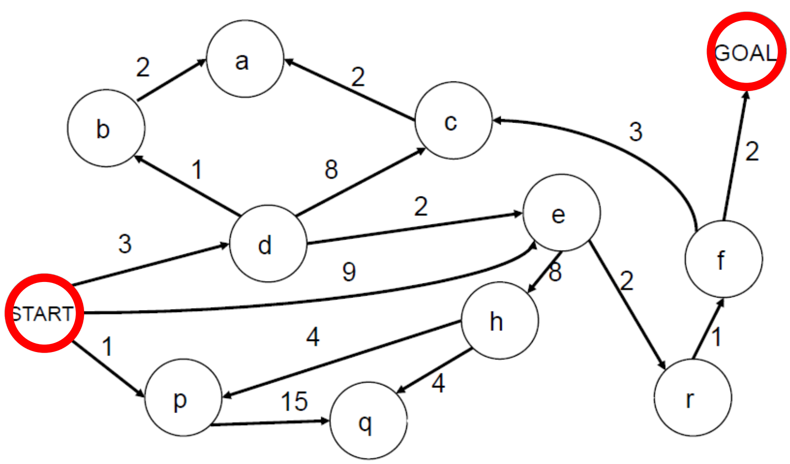
Uni



https://www.youtube.com/watch?v=XyoucHYKYSE

89

Uniform-cost search example



Expansion order:

(S,p,d,b,e,a,r,f,e,G)

90

Depth-First Iterative Deepening (DFID)

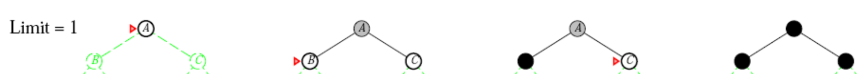
1. DFS to depth 0 (i.e., treat start node as having no successors)
2. Iff no solution, do DFS to depth 1

```
until solution found do:
  DFS with depth cutoff c;
  c = c+1
```

- **Complete**
- **Optimal/Admissible** if all operators have the same cost
  - Otherwise, not optimal, but guarantees finding solution of shortest length
- **Time complexity** is a little worse than BFS or DFS
- **Nodes near the top of the tree are generated multiple times**
  - Because most nodes are near the bottom of a tree, worst case time complexity is still exponential, O(bd)

92

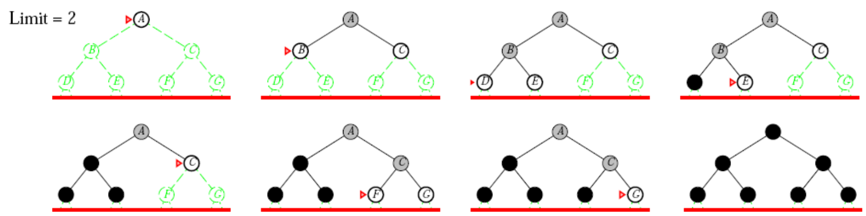
Iterative deepening search (c=1)



Nodes visited: 3

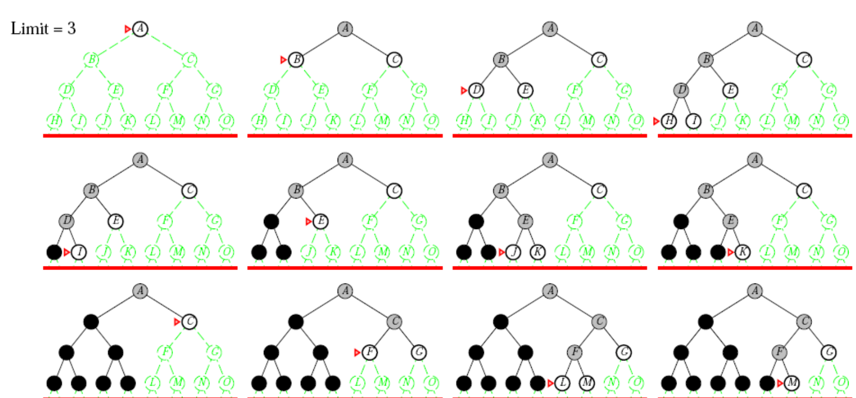
93

Iterative deepening search (c=2)



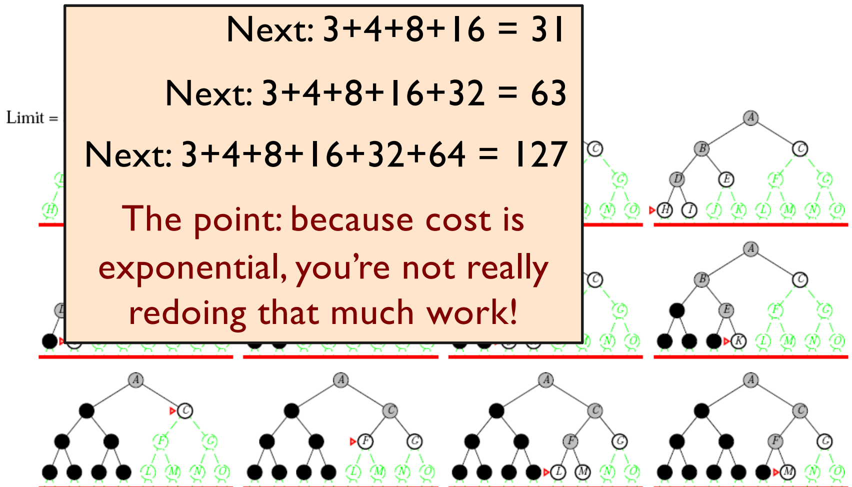
Nodes visited: 3+4 = 7

Iterative deepening search (c=3)



Nodes visited: 3+4+8 = 15

Iterative deepening search (c=3)



Nodes visited: 3+4+8 = 15

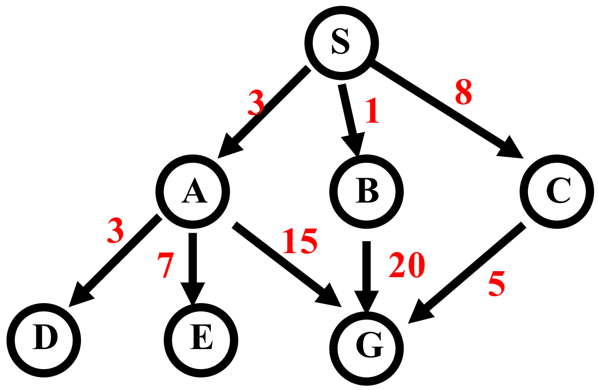
96

Depth-First Iterative Deepening

- If branching factor is *b* and solution is at depth *d*, then nodes at depth *d* are generated once, nodes at depth *d*-1 are generated twice, etc.
  - Hence $b^d + 2b^{(d-1)} + ... + db \leq b^d / (1 - 1/b)^2 = O(b^d)$.
  - If b=4, then worst case is $1.78 * 4^d$, i.e., 78% more nodes searched than exist at depth d (in the worst case).
- **Linear space complexity**, O(bd), like DFS
- Has advantage of both BFS (completeness) and DFS (limited space, finds longer paths more quickly)
- Generally preferred for **large state spaces** where **solution depth is unknown**

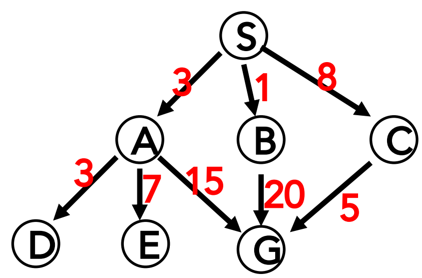
97

Example for Illustrating Search Strategies



Depth-First Search

Expanded node	Nodes list
	{ S^0 }
S^0	{ A^3 B^1 C^8 }
A^3	{ D^6 E^{10} G^{18} B^1 C^8 }
D^6	{ E^{10} G^{18} B^1 C^8 }
E^{10}	{ G^{18} B^1 C^8 }
G^{18}	{ B^1 C^8 }

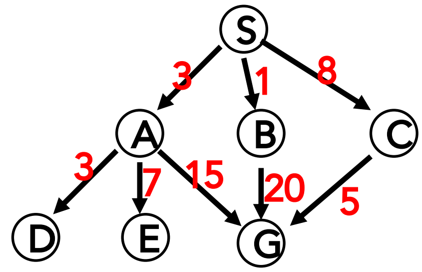


Solution path found is S A G, cost 18

Number of nodes expanded (including goal node) = 5

Breadth-First Search

Expanded node	Nodes list
	{ S^0 }
S^0	{ A^3 B^1 C^8 }
A^3	{ B^1 C^8 D^6 E^{10} G^{18} }
B^1	{ C^8 D^6 E^{10} G^{18} G^{21} }
C^8	{ D^6 E^{10} G^{18} G^{21} G^{13} }
D^6	{ E^{10} G^{18} G^{21} G^{13} }
E^{10}	{ G^{18} G^{21} G^{13} }
G^{18}	{ G^{21} G^{13} }



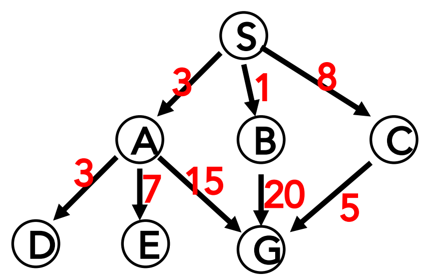
Solution path found is S A G , cost 18

Number of nodes expanded (including goal node) = 7

100

Uniform-Cost Search

Expanded node	Nodes list
	{ S^0 }
S^0	{ B^1 A^3 C^8 }
B^1	{ A^3 C^8 G^{21} }
A^3	{ D^6 C^8 E^{10} G^{18} G^{21} }
D^6	{ C^8 E^{10} G^{18} G^1 }
C^8	{ E^{10} G^{13} G^{18} G^{21} }
E^{10}	{ G^{13} G^{18} G^{21} }
G^{13}	{ G^{18} G^{21} }



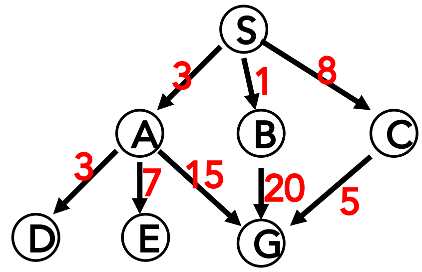
Solution path found is S C G, cost 13

Number of nodes expanded (including goal node) = 7

101

How they Perform

- **Depth-First Search:**
  - Expanded nodes: S A D E G
  - Solution found: S A G (cost 18)
- **Breadth-First Search:**
  - Expanded nodes: S A B C D E G
  - Solution found: S A G (cost 18)
- **Uniform-Cost Search:**
  - Expanded nodes: S A D B C E G
  - Solution found: S C G (cost 13)
  - *This is the only uninformed search that worries about costs.*
- **Iterative-Deepening Search:**
  - nodes expanded: S S A B C S A D E G
  - Solution found: S A G (cost 18)



102

Comparing Search Strategies

	Complete	Optimal	Time complexity	Space complexity
Breadth first search:	yes	yes	$O(b^d)$	$O(b^d)$
Depth first search	no	no	$O(b^m)$	$O(bm)$
Depth limited search	if l >= d	no	$O(b^l)$	$O(bl)$
depth first iterative deepening search	yes	yes	$O(b^d)$	$O(bd)$
bi-directional search	yes	yes	$O(b^{d/2})$	$O(b^{d/2})$

b is branching factor, d is depth of the shallowest solution,
m is the maximum depth of the search tree, l is the depth limit

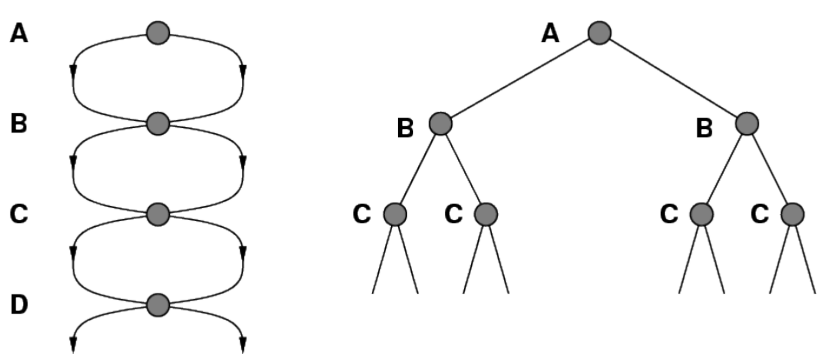
103

Avoiding Repeated States

- Ways to reduce size of state space (with increasing computational costs)
- In increasing order of effectiveness:
  1. Do not return to the state you just came from.
  2. Do not create paths with cycles in them.
  3. Do not generate any state that was ever created before.
- Effect depends on frequency of loops in state space.
  - Worst case, storing as many nodes as exhaustive search!

104

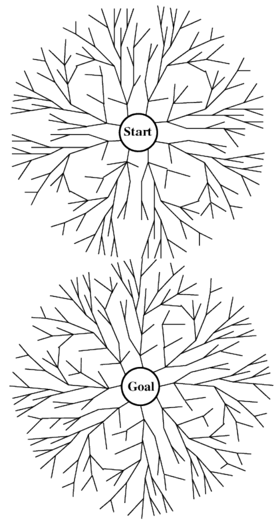
State Space → An Exponentially Growing Search Space



105

Bi-directional Search

- Alternate searching from
  - start state → goal
  - goal state → start
- Stop when the frontiers intersect.
- Works well only when there are unique start and goal states
- Requires ability to generate "predecessor" states.
- Can (sometimes) find a solution fast

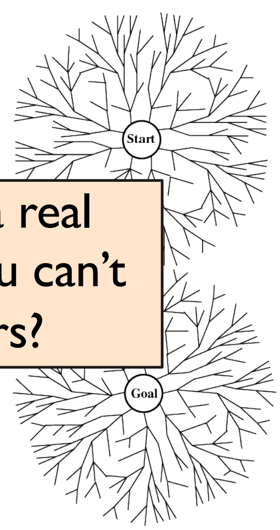


106

Bi-directional Search

- Alternate searching from
  - start state → goal
  - goal state → start
- Stop when the frontiers intersect.
- Works we
  unique st
- Requires
  "predecessor" states.
- Can (sometimes) find a solution fast

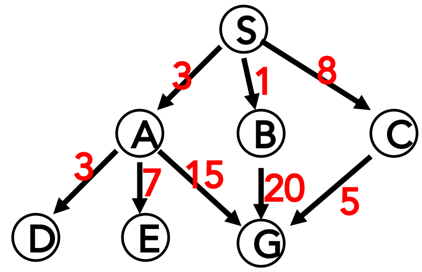
For next time: What's a real world problem where you can't generate predecessors?



107

Holy Grail Search

Expanded node	Nodes list
	{ S^0 }
S^0	{ C^8 A^3 B^1 }
C^8	{ G^{13} A^3 B^1 }
G^{13}	{ A^3 B^1 }



Solution path found is S C G, cost 13 (optimal)

Number of nodes expanded (including goal node) = 3

(minimum possible!)

108

Holy Grail Search

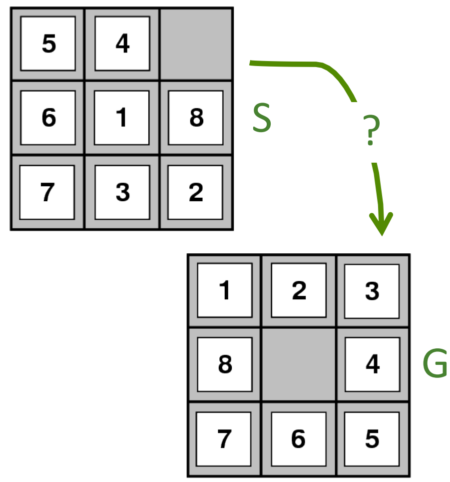
- Why not go straight to the solution, without any wasted detours off to the side?
- If we knew where the solution was we wouldn't be searching!

If only we knew where we were headed…

109

8-Puzzle Revisited

- What's a good algorithm?
  - Depth-first search?
  - Breadth-first search?
  - Uniform-cost?
  - Iterative deepening?



110

"Satisficing"

- Wikipedia: "**Satisficing** is … searching until an **acceptability threshold** is met"
- Contrast with **optimality**
  - Satisficable problems *do not get more benefit from* finding an optimal solution
- Ex: You have an A in the class. Studying for four hours will get you a 98 on the final. Studying for eight hours will get you a 100 on the final. What to do?
- A combination of *satisfy* and *suffice*
- Introduced by Herbert A. Simon in 1956

Another piece of **problem definition**

111