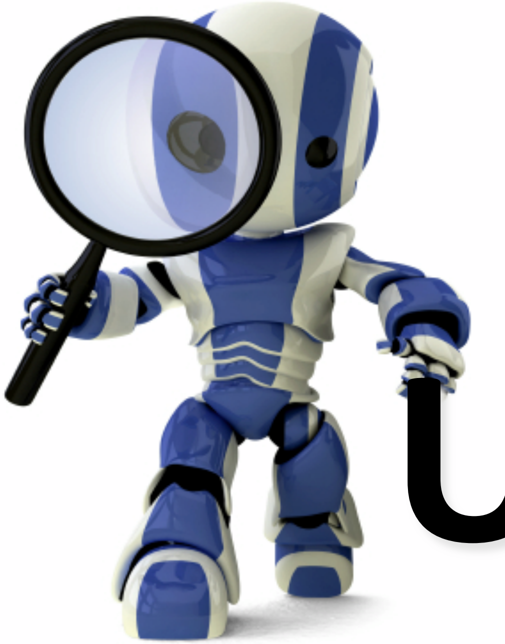


Uninformed Search

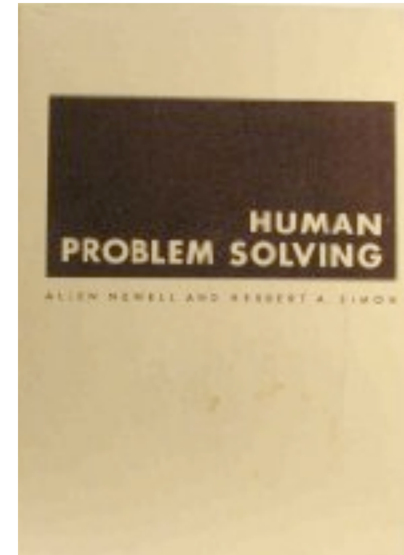
Chapter 3

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

Today's topics

- Goal-based agents
- Representing states and operators
- Example problems
- Generic state-space search algorithm
- Specific algorithms
  - Breadth-first search
  - Depth-first search
  - Uniform cost search
  - Depth-first iterative deepening
- Example problems revisited

Big Idea



Allen Newell and Herb Simon developed the *problem space principle* as an AI approach in the late 60s/early 70s

"The rational activity in which people engage to solve a problem can be described in terms of (1) a set of **states** of knowledge, (2) **operators** for changing one state into another, (3) **constraints** on applying operators and (4) **control** knowledge for deciding which operator to apply next."

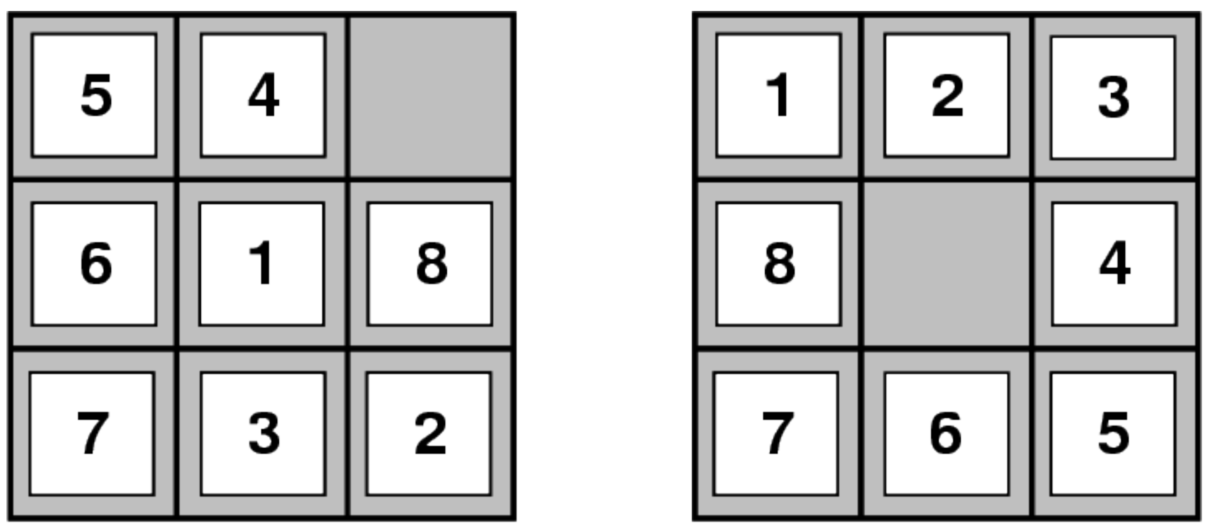
Newell A & *Simon* H A. *Human problem solving*. Englewood Cliffs, NJ: Prentice-Hall. 1972.

BTW

- Herb Simon was a polymath who contributed to economics, cognitive science, management, computer science and many other fields
- He was awarded a Nobel Prize in 1978 “for his pioneering research into the decision-making process within economic organizations”
- He is the only computer scientist to have won a Nobel Prize

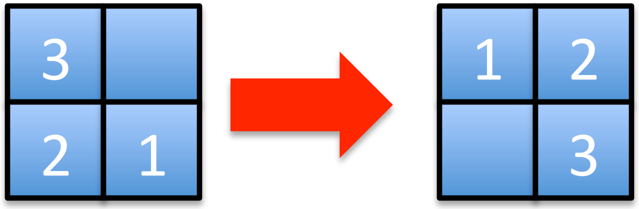
Example: 8-Puzzle

Given an initial configuration of 8 numbered tiles on a 3x3 board, move the tiles in such a way so as to produce a desired goal configuration of the tiles.



Start State Goal State

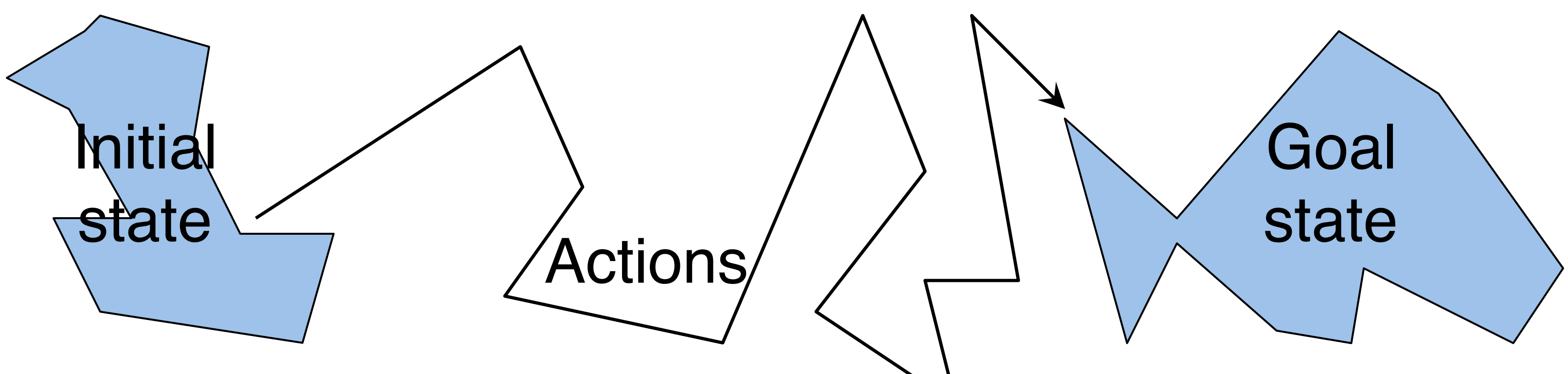
Simpler: 3-Puzzle



Building goal-based agents

We need to answer the following questions:

- How do we represent the **state** of the "world"?
- What is the **goal** to be achieved and how can re recognize it
- What are the **actions**?
- What *relevant* information should be encoded to describe the state and available transitions, and solve the problem?



What is the goal to be achieved?

- Can describe a situation we want to achieve, a set of properties that we want to hold, etc.
- Requires defining a **goal test,** so we know what it means to have achieved/satisfied goal
- A hard question, rarely tackled in AI; usually assume system designer or user specifies goal
- Psychologists and motivational speakers stress importance of establishing clear goals as a first step towards solving a problem
- What are your goals???

What are the actions?

- Characterize **primitive actions** for making changes in the world to achieve a goal
- **Deterministic** world: no uncertainty in an action's effects
- Given action and description of **current world state**, action completely specifies
  - Whether action ***can*** be applied to the current world (i.e., is it applicable and legal?) and
  - What state ***results*** after action is performed in the current world (i.e., no need for *history* information to compute the next state)

Representing actions

- Actions can be considered as **discrete events** that occur at an **instant of time**, e.g.:

  If "In class" and perform action "go home," then next state is "at home." There's no time where you're neither in class nor at home (i.e., in the state of "going home")

- Number of actions/operators depends on the **representation** used in describing a state
  - 8-puzzle: specify 4 possible moves for each of the 8 tiles, resulting in a total of **4*8=32 operators**
  - Or, we could specify four moves for "blank" square and we only need **4 operators**
- **Representational shift can simplify a problem!**

Representing states

- What information is necessary to describe all relevant aspects to solving the goal?
- The **size of a problem** is usually described in terms of the possible **number of states**
  - 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.
  - Theorem provers may deal with an infinite space
- State space size ≈ solution difficulty

Closed World Assumption

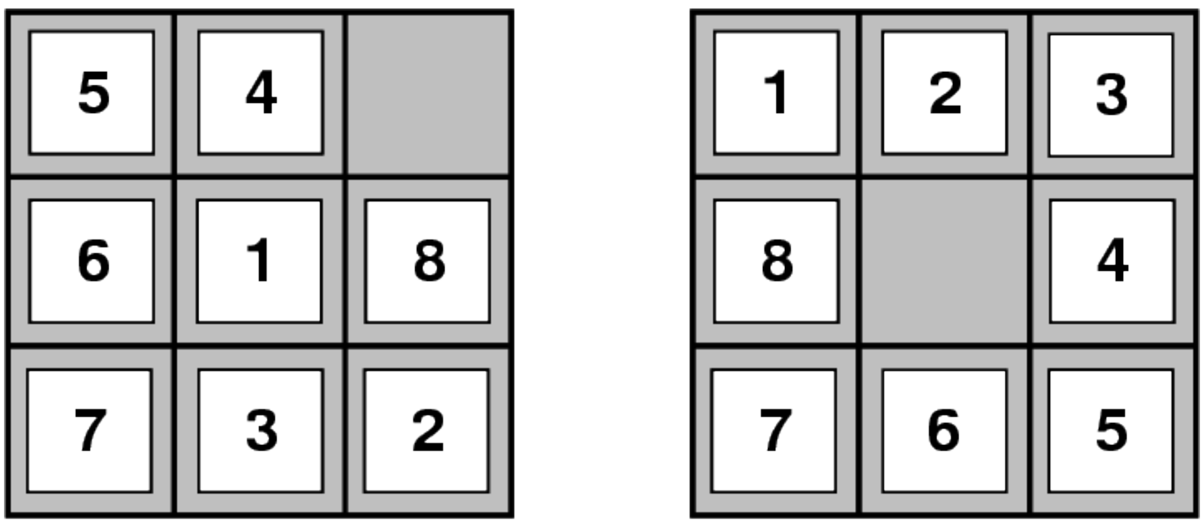
- We will generally use the **Closed World Assumption**
- All necessary information about problem domain is available in each percept so each state is a complete description of the world
- I.e., no incomplete information at any point in time

Some example problems

- Toy problems and micro-worlds
  - 8-Puzzle
  - Missionaries and Cannibals
  - Cryptarithmetic
  - Remove 5 Sticks
  - Water Jug Problem
- Real-world problems

8-Puzzle

Given an initial configuration of 8 numbered tiles on a 3x3 board, move the tiles in such a way so as to produce a desired goal configuration of the tiles.



Start State   Goal State

What are the states, goal test, actions?

8 puzzle

- **State:** 3x3 array of the tiles on the board
- **Operators:** Move blank square Left, Right, Up or Down

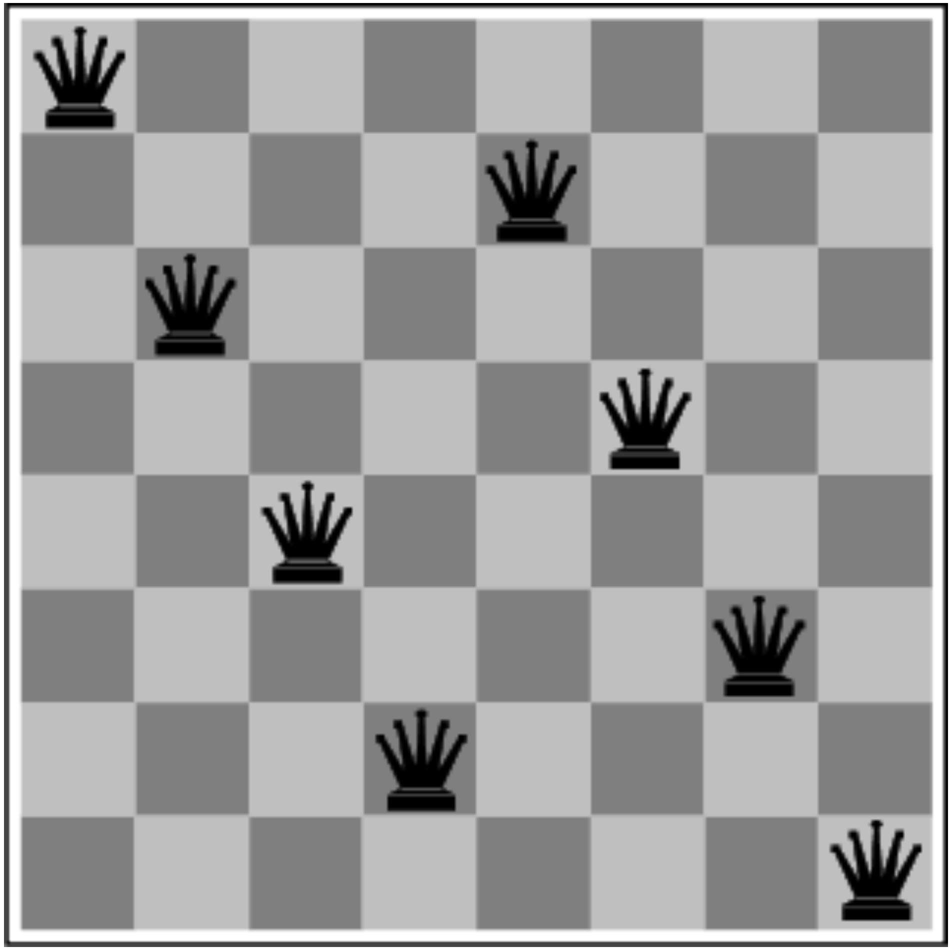
  More efficient operator encoding than one with 4 possible moves for each of 8 distinct tiles
- **Initial State:** A given board configuration
- **Goal:** A given board configuration

15 puzzle

- Popularized, but not invented by, Sam Loyd
- In late 1800s he offered $1000 to all who could find a solution
- He sold many puzzles
- The states form two disjoint spaces
- There was no path to the solution from his initial state!

The 8-Queens Puzzle

Place eight queens on a chessboard such that no queen attacks any other

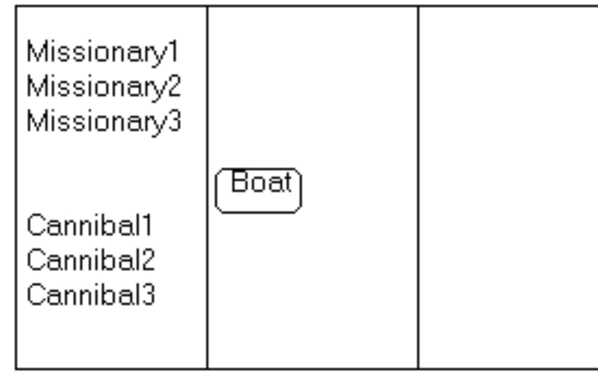


What are the states, goal test, actions?

Missionaries and Cannibals

There are 3 missionaries, 3 cannibals, and 1 boat that can carry up to two people on one side of a river

- **Goal**: Move all the missionaries and cannibals across the river
- **Constraint:** Missionaries can't be outnumbered by cannibals on either side of river, or else the missionaries are killed
- **State:** configuration of missionaries and cannibals and boat on each side of river
- **Operators:** Move boat containing some set of occupants across the river (in either direction) to the other side



3 Missionaries and 3 Cannibals wish to cross the river. They have a boat that will carry two people. Everyone can navigatethe boat. If at any time the Cannibals outnumber the missionaries on either bank of the river, they will eat the Missionaries. Find the smallest number of crossings that will allow everyone to cross the river safely.
The problem can be solved in 11 moves. But people rarely get the optimal solution, because the MC problem contains a 'tricky' state at the end, where two people move back across the river.

HW2: *What are the states, goal test, actions?*

Missionaries and Cannibals Solution

	Near side			Far side
0 Initial setup:	MMMCCC	B		-
1 Two cannibals cross over:	MMMC		B	CC
2 One comes back:	MMMCC	B		C
3 Two cannibals go over again:	MMM		B	CCC
4 One comes back:	MMMC	B		CC
5 Two missionaries cross:	MC		B	MMCC
6 A missionary & cannibal return:	MMCC	B		MC
7 Two missionaries cross again:	CC		B	MMMC
8 A cannibal returns:	CCC	B		MMM
9 Two cannibals cross:	C		B	MMMCC
10 One returns:	CC	B		MMMC
11 And brings over the third:	-		B	MMMCCC

Cryptarithmetic

- Find an assignment of digits (0..9) to letters so that a given arithmetic expression is true. Examples: SEND + MORE = MONEY and

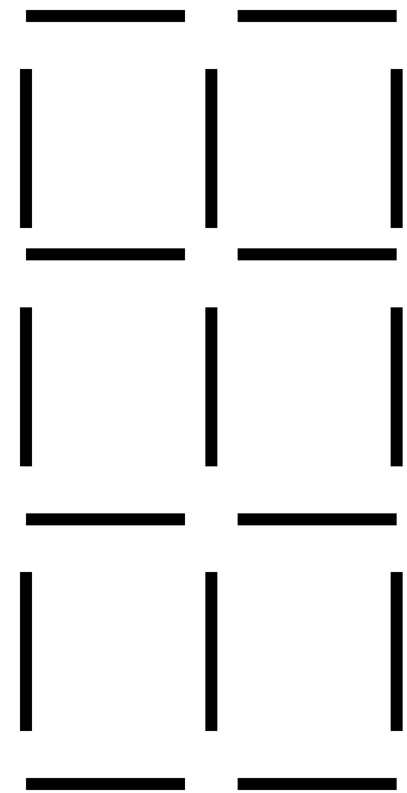
```
 FORTY     Solution:  29786
+  TEN                  850
+  TEN                  850
 -----                -----
 SIXTY                31486
F=2, O=9, R=7, etc.
```

- The solution is NOT a sequence of actions that transforms the initial state into the goal state
- Solution is a node with an assignment of digits to each of the distinct letters in the given problem

What are the states, goal test, actions?

Remove 5 Sticks

Given this configuration of sticks, remove exactly five sticks so that the remaining ones form exactly three squares



Other tasks:

- Remove 4 sticks and leave 4 squares
- Remove 3 sticks and leave 4 squares
- Remove 4 sticks and leave 3 squares

Water Jug Problem

Given full 5 gallon jug and an empty 2 gallon jug, goal is to fill the 2 gallon jug with exactly one gallon

- State = (x,y), where x is water in the 5G jug and y is water in the 2G gallon jug
- Initial State = (5,0)
- Goal State = (*,1), where * means any amount

Operator table

Name	Cond.	Transition	Effect
Empty5		$(x,y) \rightarrow (0,y)$	Empty 5G jug
Empty2		$(x,y) \rightarrow (x,0)$	Empty 2G jug
2to5	$x \leq 3$	$(x,2) \rightarrow (x+2,0)$	Pour 2G into 5G
5to2	$x \geq 2$	$(x,0) \rightarrow (x-2,2)$	Pour 5G into 2G
5to2part	$y < 2$	$(1,y) \rightarrow (0,y+1)$	Pour partial 5G into 2G

Some more real-world problems

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

Knowledge representation issues

- What's in a **state**?
  - Is boat color relevant to solving the M&C problem? Is sunspot activity relevant to predicting the stock market? This a hard problem that's usually left to a person.
- The right **level of abstraction** to describe the world
  - Too fine-grained and we'll "miss the forest for the trees." Too coarse-grained and we'll miss critical details for solving the problem.
- Number of states depends on the representation and abstraction level. E.g., for Remove-5-Sticks
  - Represent individual sticks, there are 17-choose-5 possible ways of removing 5 sticks
  - Represent the "squares" defined by 4 sticks, there are 6 squares initially and we must remove 3 squares, so only 6-choose-3 ways of removing 3 squares

Formalizing search in a state space

- A state space is a **graph** (V, E) where V is a set of **nodes** and E is a set of **arcs**, and each arc is directed from a node to another node
- **Nodes** are data structures with a state description and other info, e.g., node's parent, name of operator that generated it from parent, etc.
- **Arcs** are instances of operators. When the operator is applied to the state at its source node, then resulting state is arc's destination node

Formalizing search in a state space

- Each arc has fixed, positive **cost** associated with it corresponding to the operator cost
- Each node has a set of **successor nodes** corresponding to all of legal actions that can be applied at node's state
  - Expanding a node = generating its successor nodes and adding them and their associated arcs to the graph
- One or more nodes are marked as **start nodes**
- A **goal test** predicate is applied to a state to determine if its associated node is a goal node

Example: Water Jug Problem

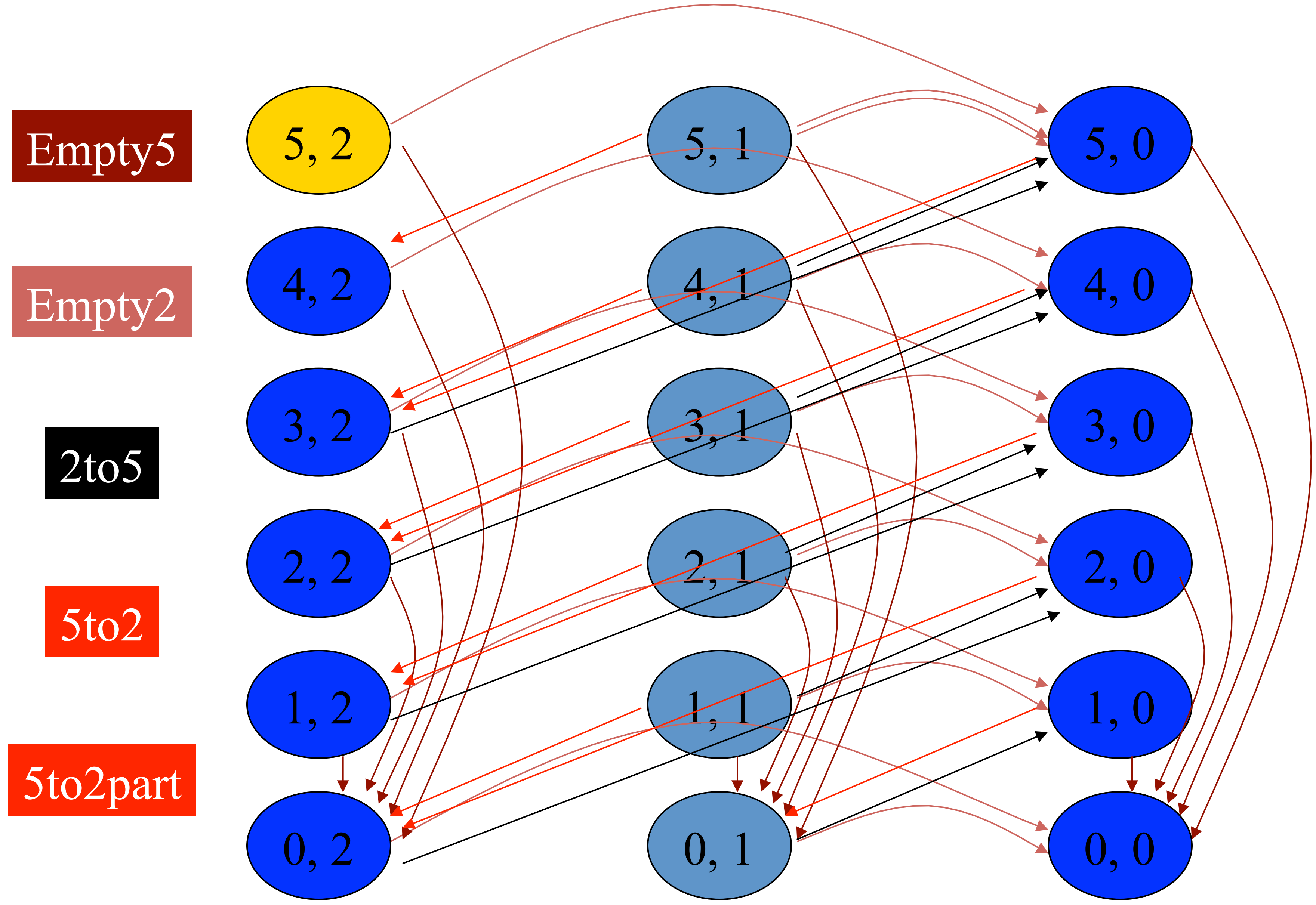
Given full 5 gallon jug and an empty 2 gallon jug, goal is to fill the 2 gallon jug with exactly one gallon

- State = (x,y), where x is water in the 5G jug and y is water in the 2G gallon jug
- Initial State = (5,0)
- Goal State = (*,1), where * means any amount

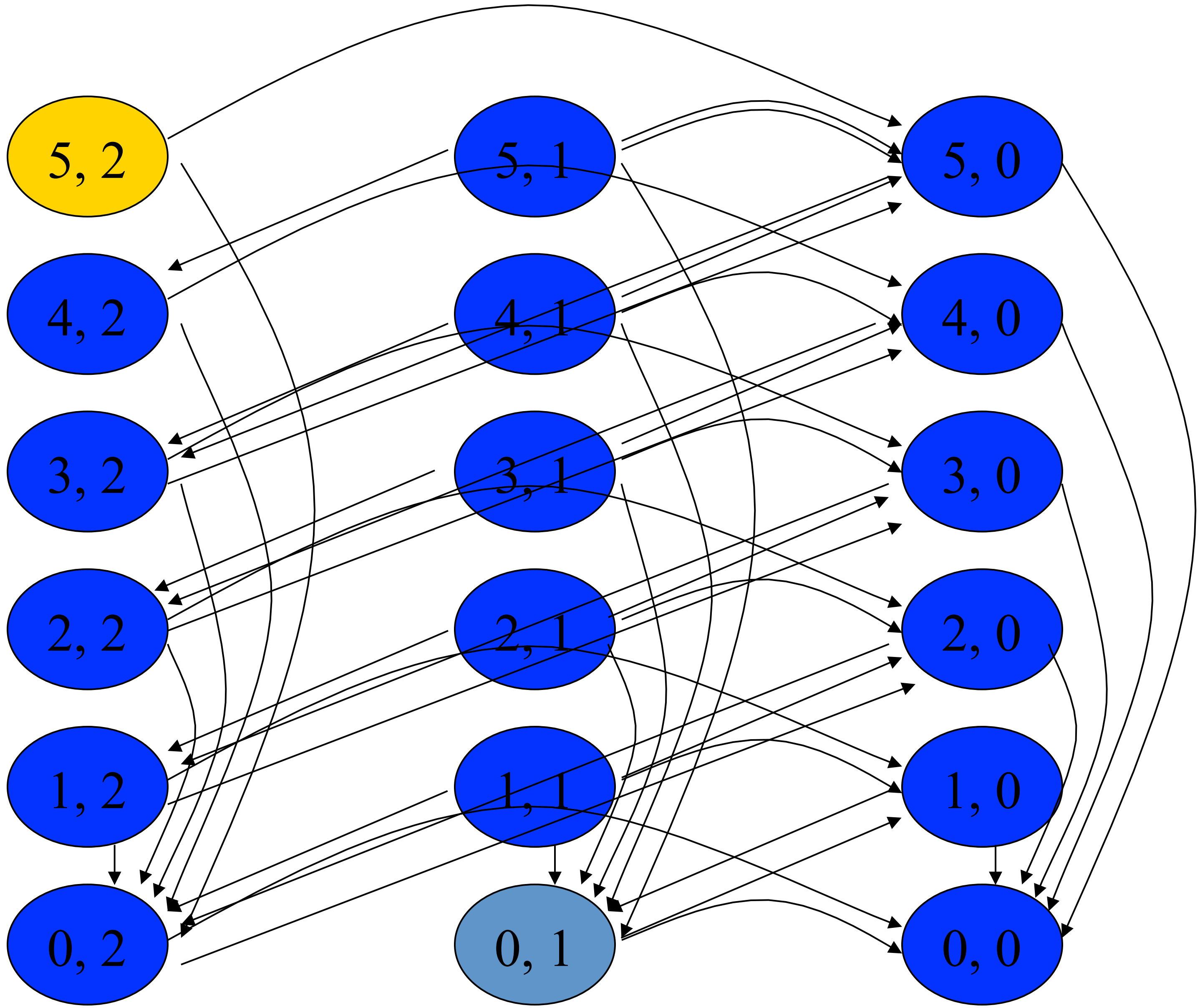
Operator table

Name	Cond.	Transition	Effect
Empty5		$(x,y) \rightarrow (0,y)$	Empty 5G jug
Empty2		$(x,y) \rightarrow (x,0)$	Empty 2G jug
2to5	$x \leq 3$	$(x,2) \rightarrow (x+2,0)$	Pour 2G into 5G
5to2	$x \geq 2$	$(x,0) \rightarrow (x-2,2)$	Pour 5G into 2G
5to2part	$y < 2$	$(1,y) \rightarrow (0,y+1)$	Pour partial 5G into 2G

Water jug state space



Water jug solution



Class Exercise

- Representing a 2x2 Sudoku puzzle as a search space
- Fill in the grid so that every row, every column, and every 2x2 box contains the digits 1 through 4.
  - What are the states?
  - What are the operators?
  - What are the constraints (on operator application)?
  - What is the description of the goal state?

	3		
			1
3			
		2	

Formalizing search (3)

- **Solution:** sequence of actions associated with a path from a start node to a goal node
- **Solution cost:** sum of the arc costs on the solution path
  - If all arcs have same (unit) cost, then solution cost is just the length of solution (number of steps / state transitions)

Formalizing search (4)

- **State-space search:** searching through state space for solution by **making explicit** a sufficient portion of an **implicit** state-space graph to find a goal node
  - Can’t materializing whole space for large problems
  - Initially V={S}, where S is the start node, E={}
  - On expanding S, its successor nodes are generated and added to V and associated arcs added to E
  - Process continues until a goal node is found
- Nodes represent a *partial solution* path (+ cost of partial solution path) from S to the node
  - From a node there may be many possible paths (and thus solutions) with this partial path as a prefix

State-space search algorithm

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

```
function general-search (problem, QUEUEING-FUNCTION)
  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

Key procedures to be defined

- EXPAND
  - Generate all successor nodes of a given node
- GOAL-TEST
  - Test if state satisfies all goal conditions
- QUEUEING-FUNCTION
  - Used to maintain a ranked list of nodes that are candidates for expansion

Bookkeeping

Typical node data structure includes:

- State at this node
- Parent node
- Operator applied to get to this node
- Depth of this node (number of operator applications since initial state)
- Cost of the path (sum of each operator application so far)

Some issues

- Search process constructs a search tree/graph, where
  - **root** is initial state and
  - **leaf nodes** are nodes
    - not yet expanded (i.e., in list "nodes") or
    - having no successors (i.e., they're *deadends* because no operators were applicable and yet they are not goals)
- Search graph may be infinite because of loops even if state space is small
- Return a *path* or a *node*, depending on problem.
  - E.g., in cryptarithmetic return a node; in 8-puzzle, a path
- Changing definition of the QUEUEING-FUNCTION leads to different search strategies

Evaluating search strategies

- **Completeness**
  - Guarantees finding a solution whenever one exists
- **Time complexity** (worst or average case)
  - Usually measured by *number of nodes expanded*
- **Space complexity**
  - Usually measured by maximum size of the graph during the search
- **Optimality/Admissibility**
  - If a solution is found, is it guaranteed to be an optimal one, i.e., one with minimum cost

Uninformed vs. informed search

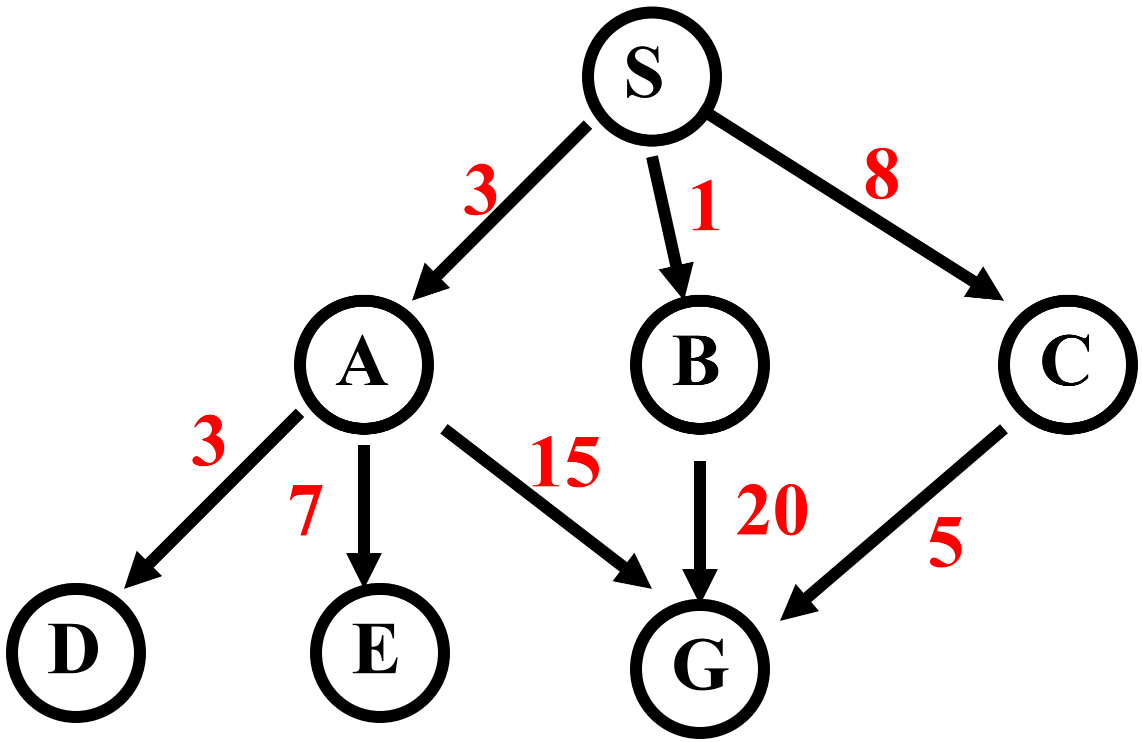
Uninformed search strategies (blind search)

- Use no information about likely "direction" of goal node(s)
- Methods: breadth-first, depth-first, depth-limited, uniform-cost, depth-first iterative deepening, bidirectional

Informed search strategies (heuristic search)

- Use information about domain to (try to) (usually) head in the general direction of goal node(s)
- Methods: hill climbing, best-first, greedy search, beam search, A, A*

Example of uninformed search strategies



Consider this search space where S is the start node and G is the goal. Numbers are arc costs.

Classic uninformed search methods

- The four classic uninformed search methods
  - Breadth first search (BFS)
  - Depth first search (DFS)
  - Uniform cost search *(generalization of BFS)*
  - Iterative deepening *(blend of DFS and BFS)*
- To which we can add another technique
  - Bi-directional search *(hack on BFS)*

Breadth-First Search

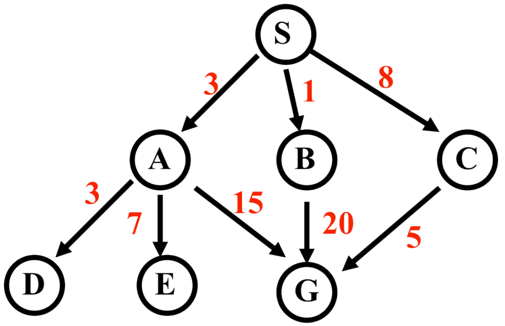
- Enqueue nodes in **FIFO** (first-in, first-out) order
- **Complete**
- **Optimal** (i.e., admissible) if all operators have same cost. Otherwise, not optimal but finds solution with shortest path length.
- **Exponential time and space complexity**, $O(b^d)$, where d is depth of the solution and b is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

– A complete search tree of depth d where each non-leaf node has b children, has total of $1 + b + b^2 + ... + b^d = (b^{(d+1)} - 1)/(b-1)$ nodes

– For a search tree of depth 12, where nodes at depths 0..11 have 10 children and nodes at depth 12 have 0, there are $1+10+100+1000...10^{12} = (10^{13}-1)/9 = O(10^{12})$ nodes

– If BFS expands 1000 nodes/sec and nodes uses 100 bytes, then BFS takes 35 years to run in the worst case, and it will use 111 terabytes of memory!

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} }

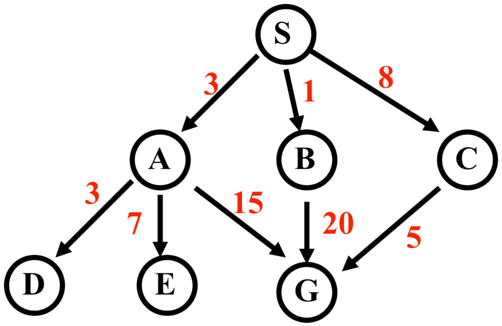
Solution path found is S A G , cost 18

Number of nodes expanded (including goal node) = 7

Depth-First (DFS)

- Enqueue nodes on nodes in **LIFO** (last-in, first-out) order, i.e., use stack data structure to order nodes
- **May 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)
- Can find **long solutions quickly** if lucky (and **short solutions slowly** if unlucky!)
- When search hits deadend, can only back up one level at a time even if “problem” occurs because of a bad choice at top of tree

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 }

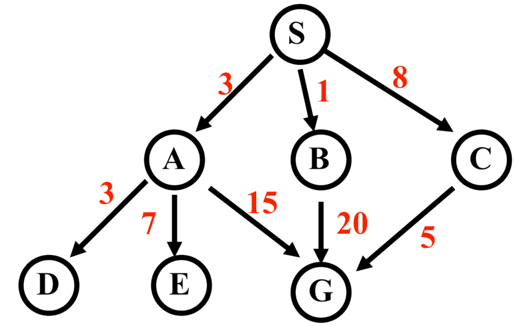
Solution path found is S A G, cost 18

Number of nodes expanded (including goal node) = 5

Uniform-Cost (UCS)

- Enqueue nodes by **path cost**. i.e., let g(n) = cost of path from start to current node n. Sort nodes by increasing value of g.
- Also called *Dijkstra's Algorithm*, similar to *Branch and Bound Algorithm* from operations research
- **Complete (*)**
- **Optimal/Admissible** (*)
  - Admissibility depends on goal test being applied *when a node is removed from nodes list*, not when its parent node is expanded and the node is first generated
- **Exponential time and space complexity**, $O(b^d)$

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^{21} }
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

Depth-First Iterative Deepening (DFID)

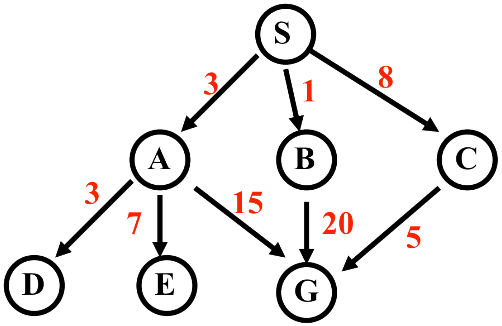
- Do DFS to depth 0, then, if no solution, do DFS to depth 1, etc.
- Usually used with a tree search
- **Complete**
- **Optimal/Admissible** if all operators have same cost, otherwise, guarantees finding solution of shortest length (like BFS)
- Time complexity a bit worse than BFS or DFS

  Nodes near top of search tree generated many times, but since almost all nodes are near tree bottom, worst case time complexity is still exponential, $O(b^d)$

Depth-First Iterative Deepening (DFID)

- 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 $b^d + 2b^{(d-1)} + ... + db <= b^d / (1 - 1/b)^2 = O(b^d)$.
  - If b=4, worst case is $1.78 * 4^d$, i.e., 78% more nodes searched than exist at depth d (in worst case)
- **Linear space complexity**, O(bd), like DFS
- Has advantages of BFS (completeness) and DFS (i.e., limited space, finds longer paths quickly)
- Preferred for **large state spaces** where **solution depth is unknown**

How they perform



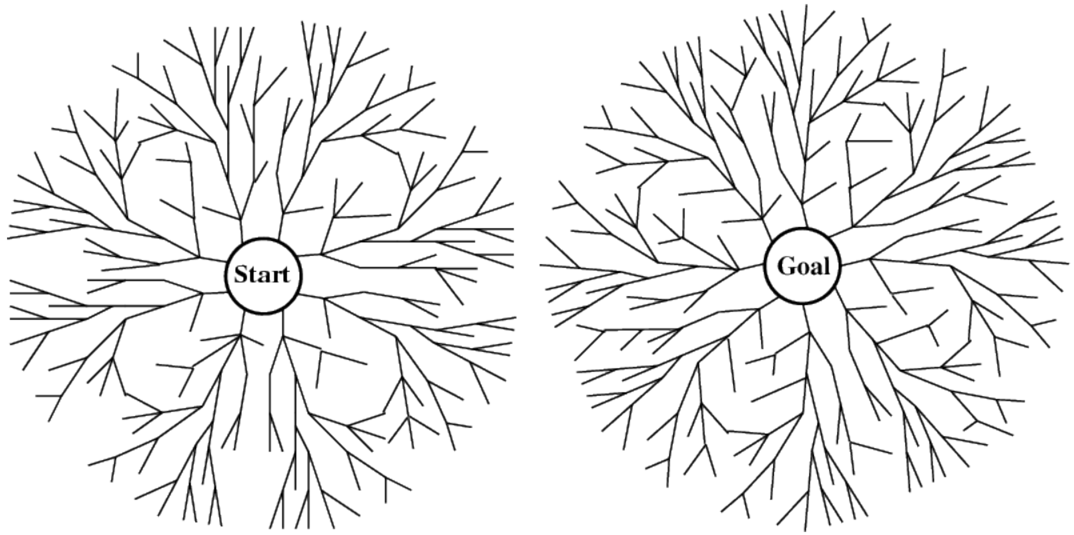
- **Depth-First Search:**
  - 4 Expanded nodes: S A D E G
  - Solution found: S A G (cost 18)
- **Breadth-First Search**:
  - 7 Expanded nodes: S A B C D E G
  - Solution found: S A G (cost 18)
- **Uniform-Cost Search**:
  - 7 Expanded nodes: S A D B C E G
  - Solution found: S C G (cost 13)

  Only uninformed search that worries about costs
- **Iterative-Deepening Search**:
  - 10 nodes expanded: S S A B C S A D E G
  - Solution found: S A G (cost 18)

Searching Backward from Goal

- Usually a successor function is reversible
  - i.e., can generate a node's predecessors in graph
- If we know a single goal (rather than a goal's properties), we could search backward to the initial state
- It might be more efficient
  - Depends on whether the graph fans in or out

Bi-directional search



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

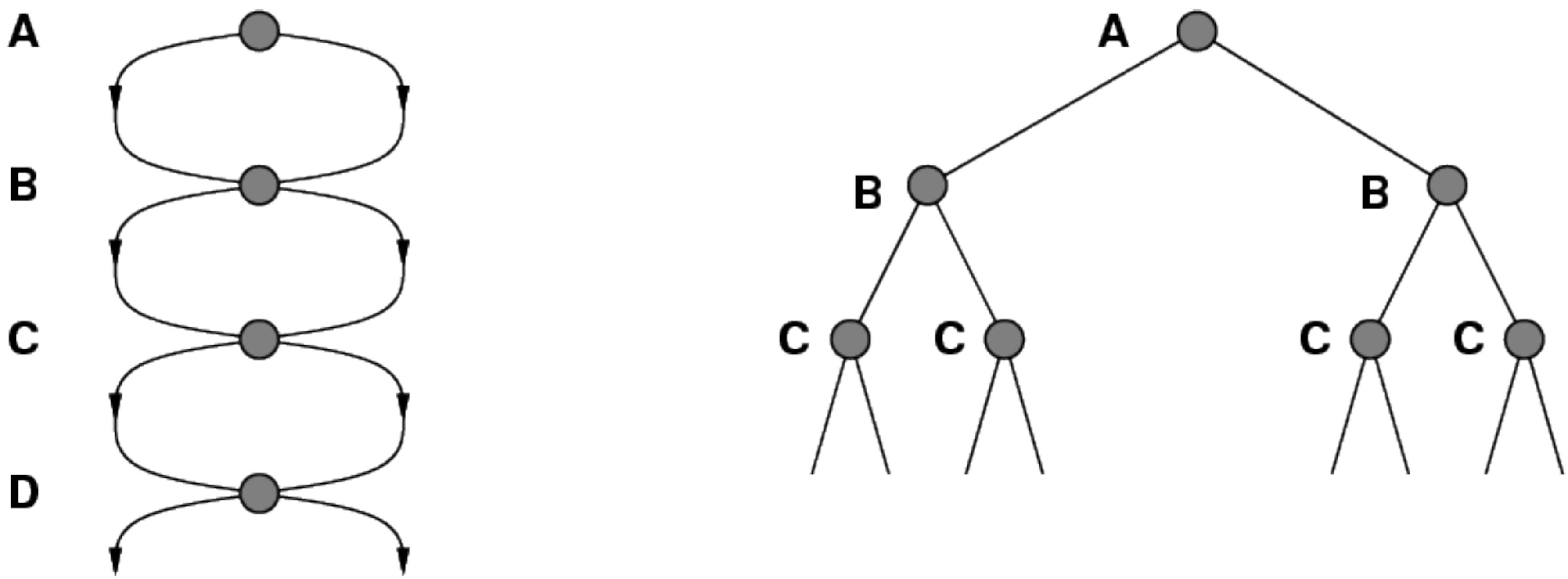
Comparing Search Strategies

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Time	b^d	b^d	b^m	b^l	b^d	$b^{d/2}$
Space	b^d	b^d	bm	bl	bd	$b^{d/2}$
Optimal?	Yes	Yes	No	No	Yes	Yes
Complete?	Yes	Yes	No	Yes, if $l \geq d$	Yes	Yes

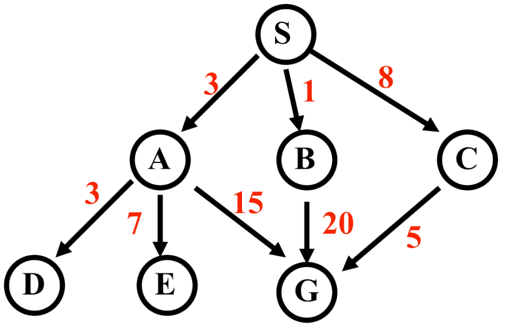
Some simple improvements

- In increasing order of effectiveness in reducing size of state space and with increasing computational costs:
    1. Never return to state you just came from
    2. Never create paths with cycles in them
    3. Never generate a state that was ever created before
- Net effect depends on frequency of *loops i*n state space

A State Space that Generates an Exponentially Growing Search Space



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

(as few as possible!)

If only we knew where we were headed…