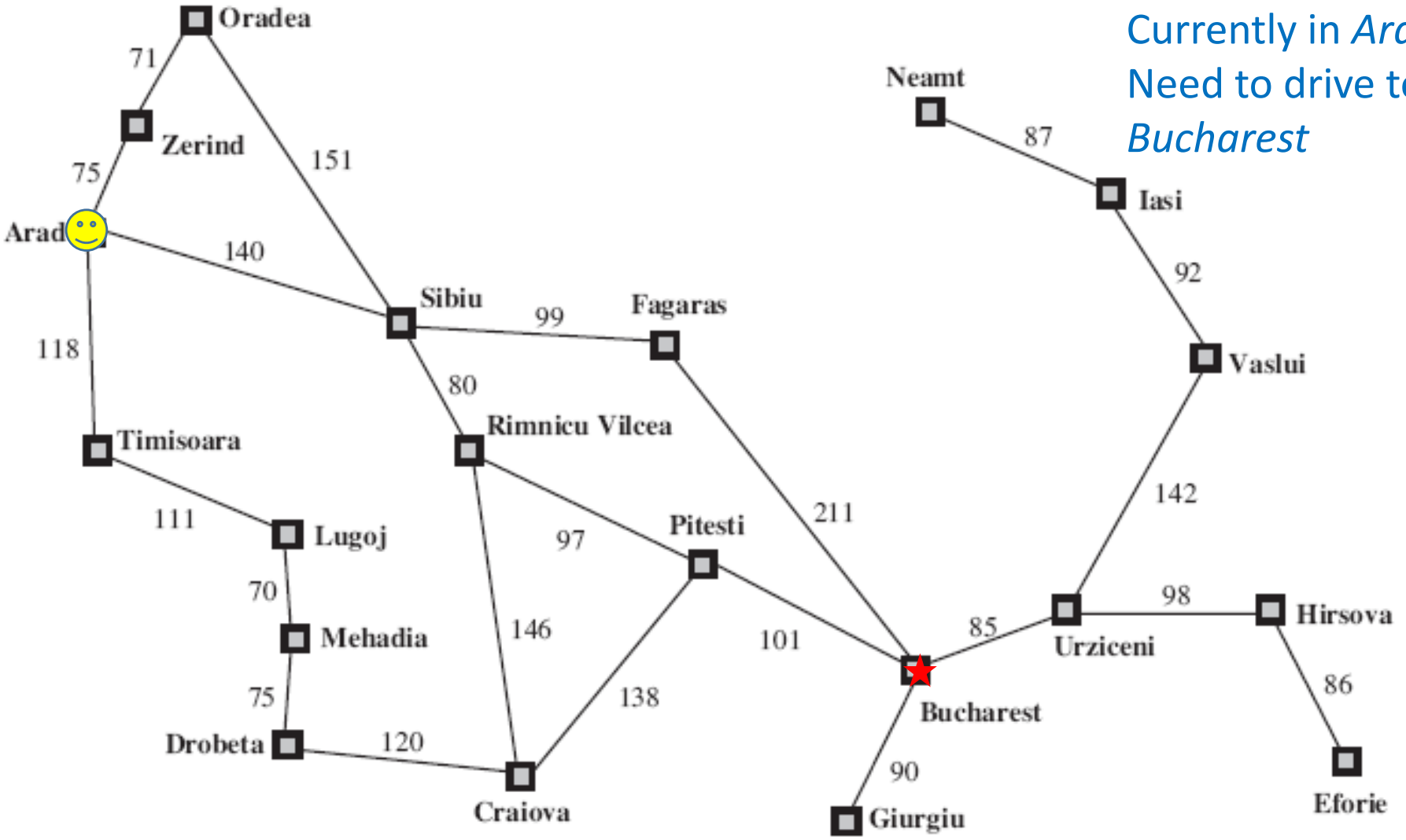
Solving Problems by Searching

COEN166

Artificial Intelligence

Navigation in Romania



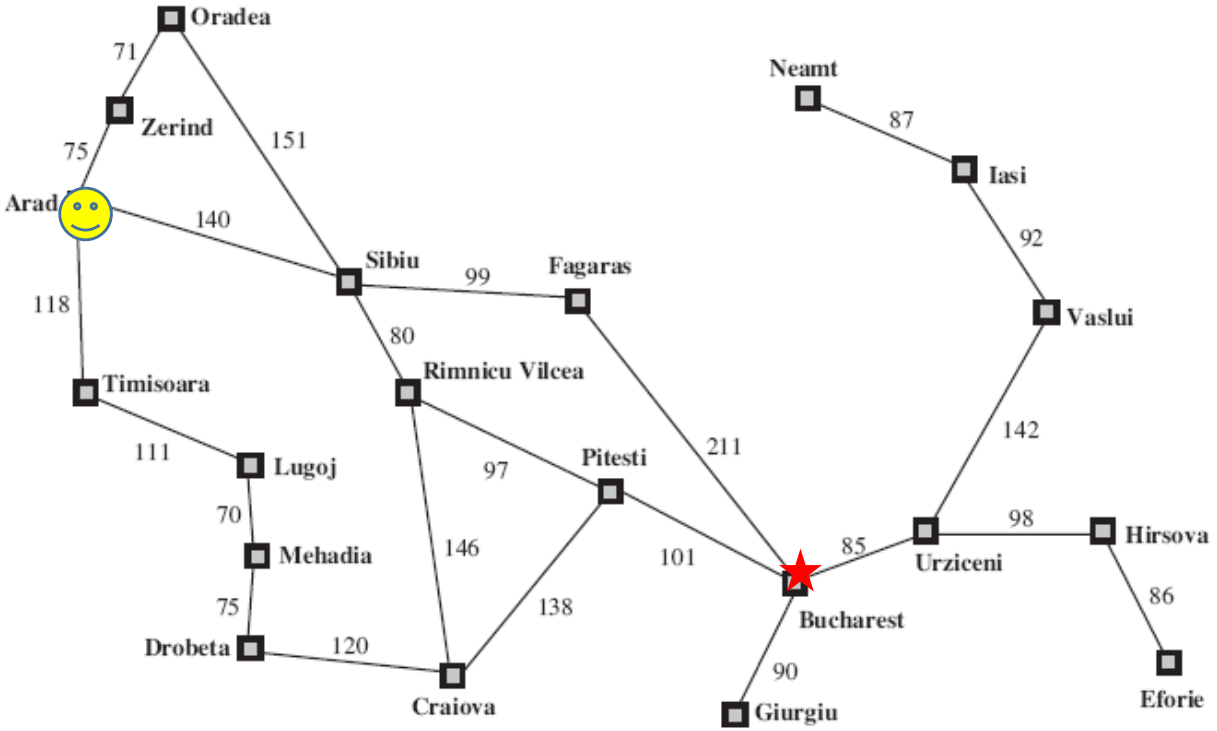
Agent:
Currently in *Arad*;
Need to drive to
Bucharest

Problem-Solving Agents

- **Goal formulation**
  - Goals help organize behavior by limiting
    - the **objectives** that the agent is trying to achieve, and hence
    - the **actions** it needs to consider
  - *Goal:* Get to Bucharest from Arad
- **Problem Formulation:** the process of deciding what actions and states to consider, given a goal
  - *Actions:* driving from one major town to another
  - *States:* being in a particular town

Problem-Solving Agents

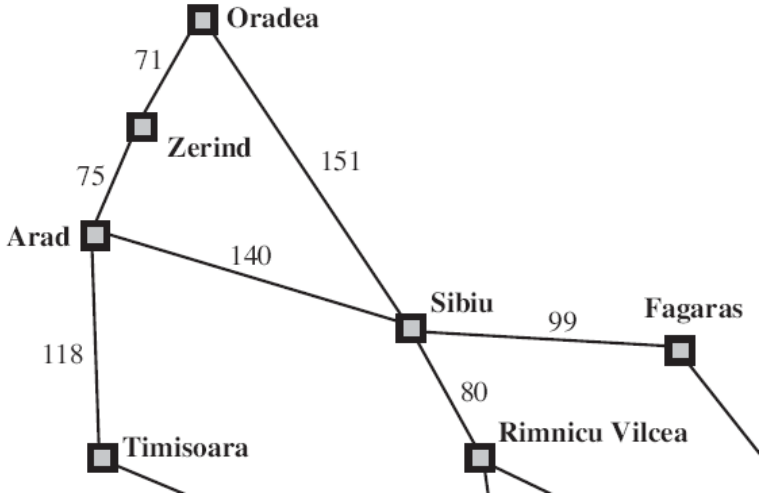
- **Search:** the process of looking for a sequence of actions that reaches the goal
  - Where to go from Arad? Sibiu, Timisoara, or Zerind?
    - Environment unknown – *trying one of the actions at random*
    - Environment observable (a map of Romania) – *decide what to do by first examining future actions*



Problem-Solving Agents

- **Search Algorithm:** takes a problem as input and returns a solution in the form of an action sequence
- **Solution:** an action sequence that leads from the initial state to a goal state
- **Execution Phase:** once a solution is found, the actions it recommends can be carried out

Well-defined Problems and Solutions



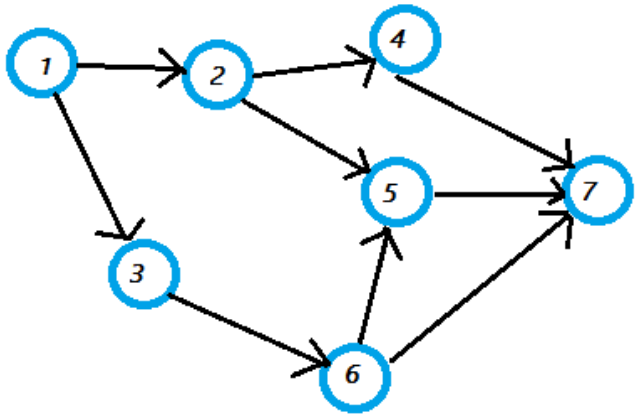
- Problem Definition - 5 components
  1. Initial State
     - ***In*(*Arad*)**
  2. Actions (available to the agent)
     - Given a state *s*, **Actions(*s*)** returns the set of actions that can be executed in *s*
     - Each of these actions is *applicable* in *s*
     - **Actions(*In*(*Arad*)): {*Go*(*Sibiu*), *Go*(*Timisoara*), *Go*(*Zerind*)}**
  3. Transition Model (a description of what each action does)
     - **Result(*s*,*a*)**: returns the state that results from doing action *a* in state *s*
     - **Result(In(Arad), Go(zerind)) = In(Zerind)**

Well-defined Problems and Solutions

- Problem Definition - 5 components

  The *State Space:* the set of all states reachable from the initial state by any sequence of actions.

  - Implicitly defined by initial state, actions, transition model
  - A directed graph (e.g. the map of Romania)
    - Nodes: the states
    - Links: actions

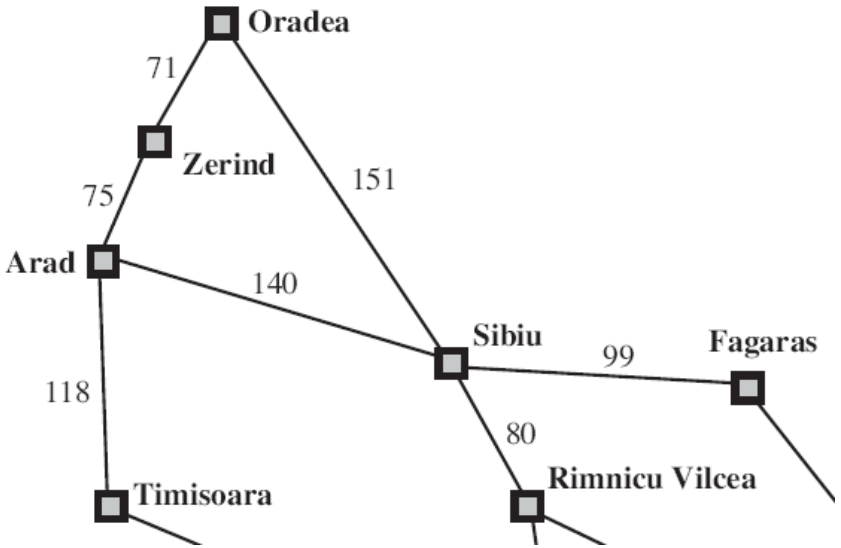


  - A *path* in the state space: a sequence of states connected by a sequence of actions

Well-defined Problems and Solutions

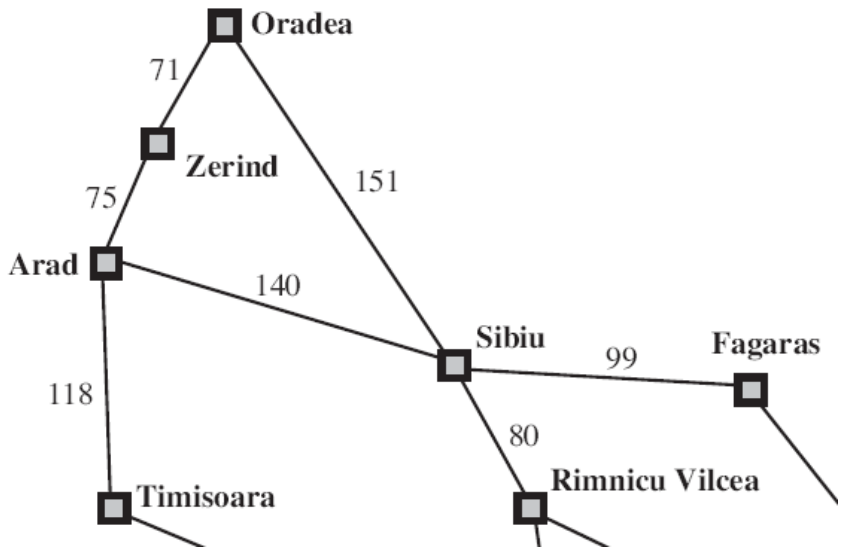
- Problem Definition - 5 components
  4. *Goal test:* determines whether a given state is a goal state
     - e.g. In the Romania Navigation problem: the goal state is the singleton set {*In*(*Bucharest*)}

  5. *Path cost function*: assigns a numeric cost to each path
     - e.g. Romania: the cost of a path can be its length in kilometers
     - step cost ***c(s,a,s')***: the cost of taking action ***a*** in state ***s*** to reach state ***s'***



Well-defined Problems and Solutions

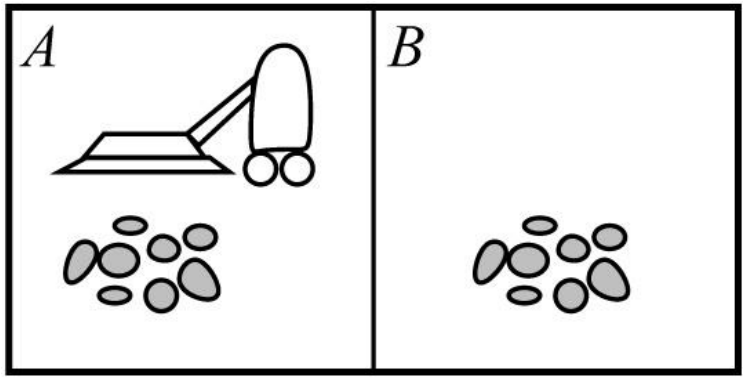
- A solution to a problem
  - An action sequence that leads from the initial state to a goal state.
  - Solution quality: measured by the path cost function
  - Optimal solution: lowest path cost



Vacuum World

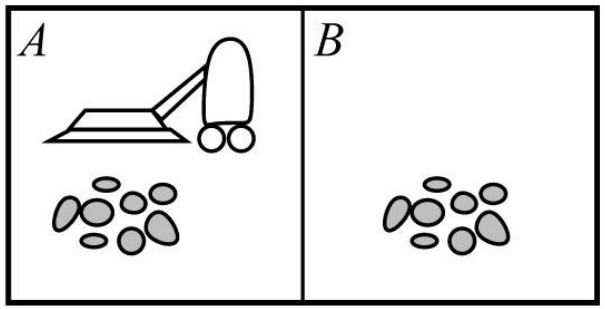
- States
  - 2 agent locations (squares), each location might or might not contain dirt
  - $2 \times 2^2 = 8$ states
  - If *n* locations, then how many states?

    $n \times 2^n$ states



Vacuum World

- Initial state
  - Any state can be designated as the initial state
- Actions
  - Left, Right, Suck
  - Larger environments: Up, Down



- Transition Model
  - Every action has an effect, except
    - Moving Left in the leftmost square,
    - Moving Right in the rightmost square,
    - Sucking in a clean square

Vacuum World

The complete state space (and transition model)

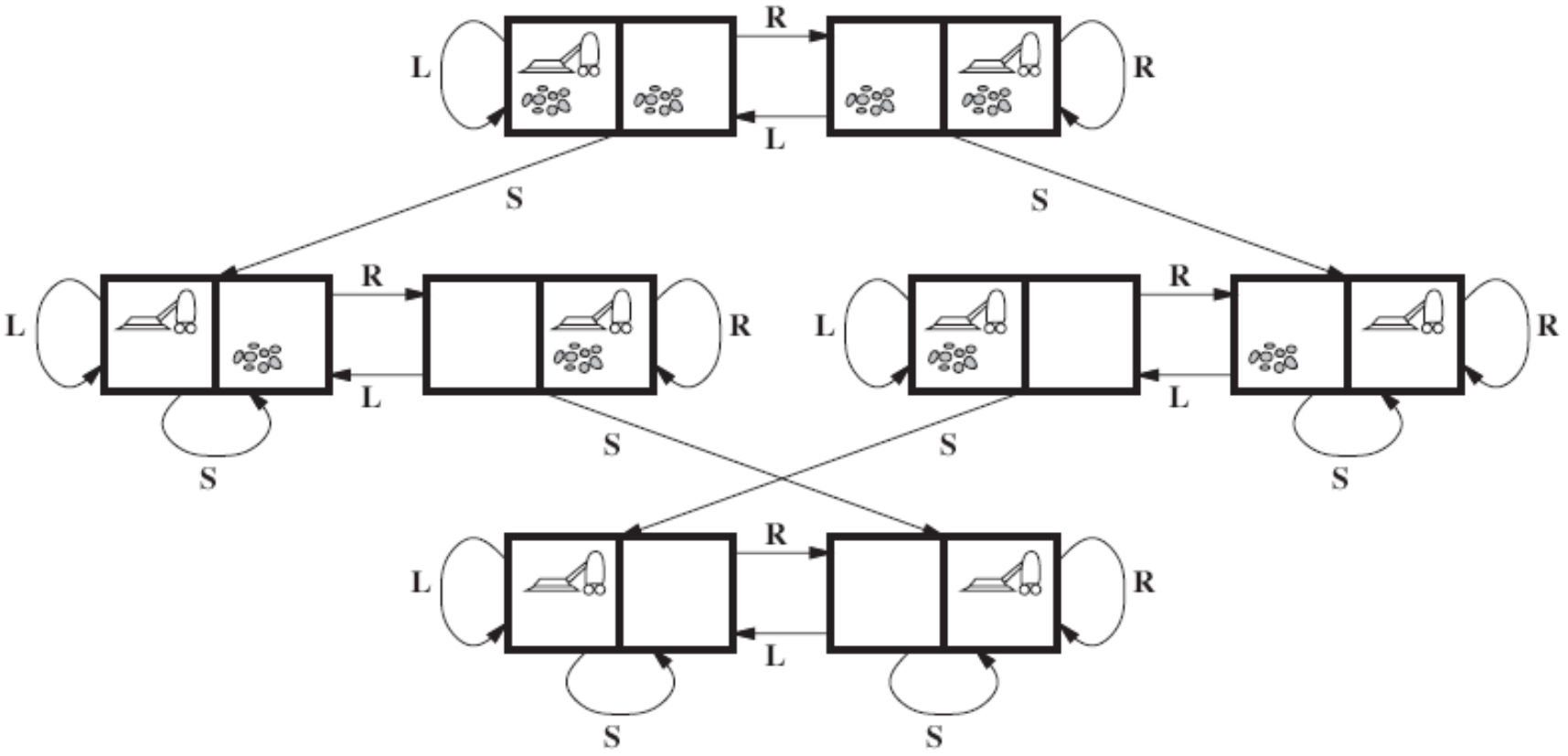


Figure 3.3 **FILES: figures/vacuum2-state-space.eps (Tue Nov 3 16:24:01 2009).** The state space for the vacuum world. Links denote actions: L = *Left*, R = *Right*, S = *Suck*.

Vacuum World

- Goal Test
  - Check whether all squares are clean
- Path cost
  - Each step costs 1 (i.e. 1 per action)
  - The path cost is the number of steps (actions) in the path

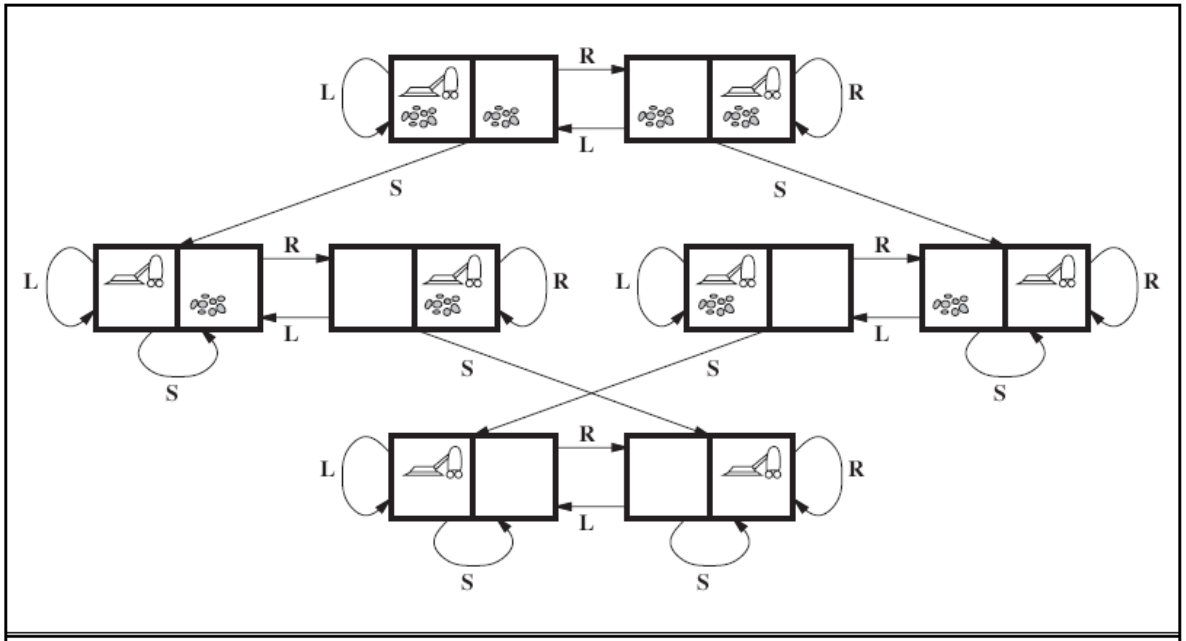
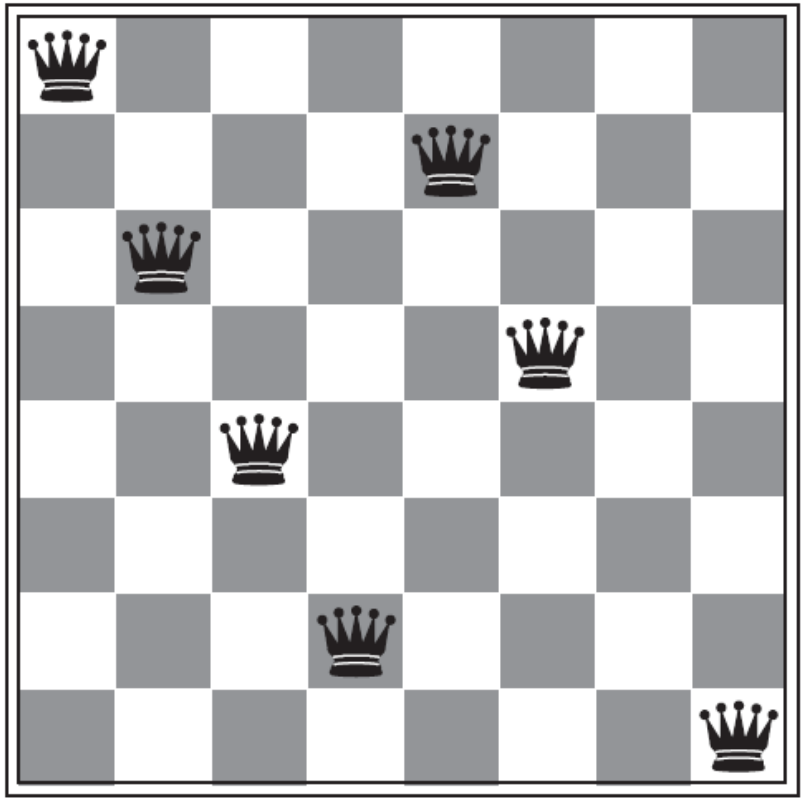


Figure 3.3 **FILES: figures/vacuum2-state-space.eps (Tue Nov 3 16:24:01 2009).** The state space for the vacuum world. Links denote actions: L = *Left*, R = *Right*, S = *Suck*.

The 8-Queens Problem

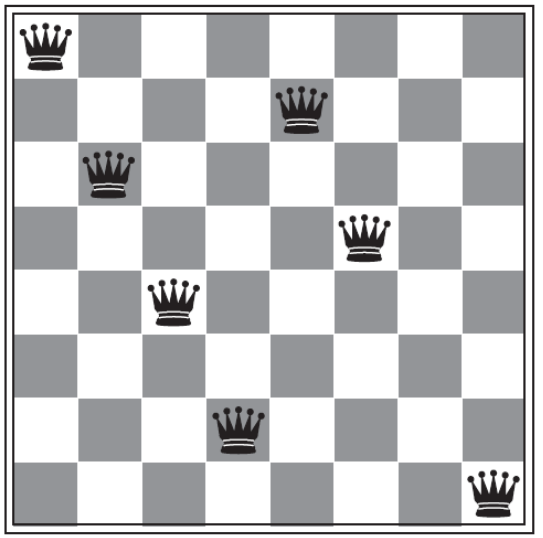
- Goal: place 8 queens on a chessboard such that no queen attacks any other
- Two queens attack each other if
  - They are in the same row, or column, or diagonal line

Almost a solution



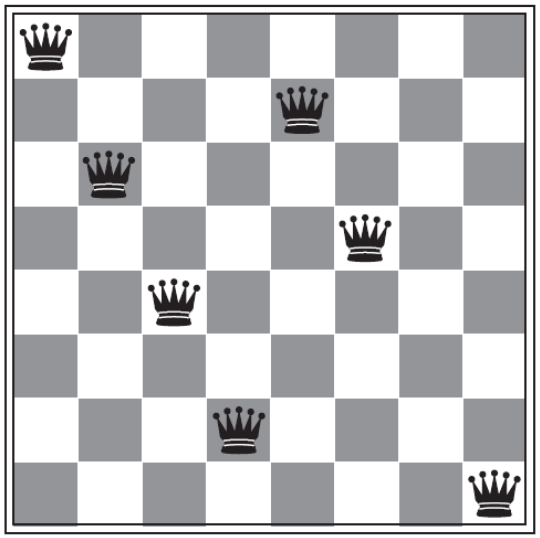
The 8-Queens Problem

- Incremental formulation: starts with an empty state, then each action adds a queen to the state
  - States?
    - any arrangement of $n \leq 8$ queens on the board
    - Or arrangements of $n \leq 8$ queens in the leftmost n columns, 1 per column, such that no queen attacks any other
  - Initial state?
    - no queens on the board
  - Actions?
    - add a queen to any empty square
    - Or add queen to leftmost empty square such that it is not attacked by other queens



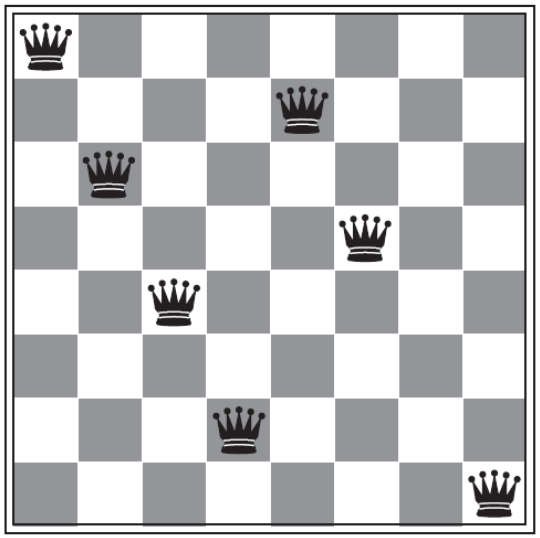
The 8-Queens Problem

- Incremental formulation: starts with an empty state, then each action adds a queen to the state
  - Transition model?
    - Returns the board with a queen added to the specified square
  - Goal test?
    - 8 queens are on the board, none attacked
  - Path cost?
    - 1 per move



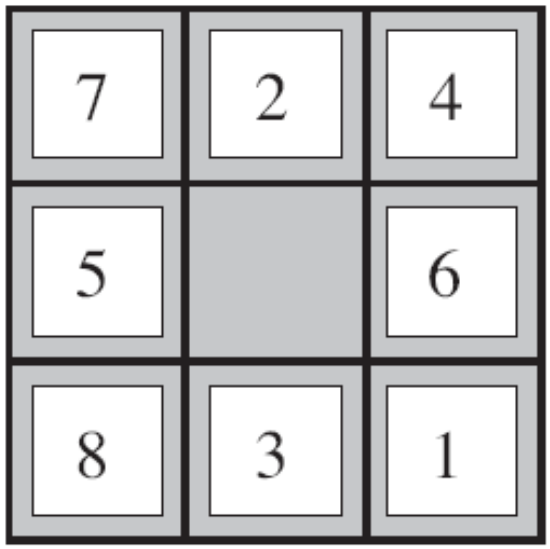
The 8-Queens Problem

- Complete-state formulation: starts with all 8 queens on the board, and moves them around

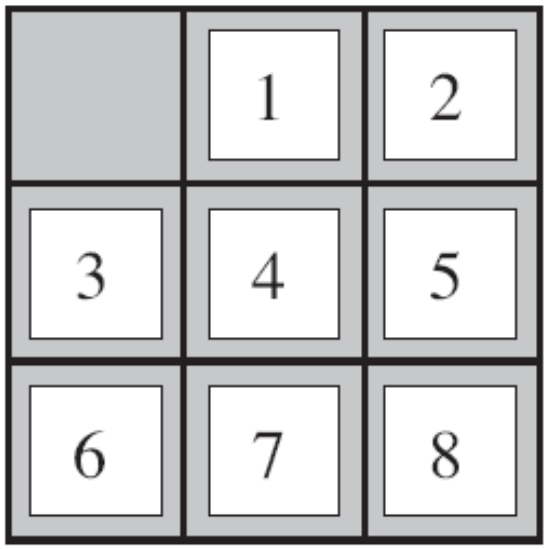


The 8-puzzle

- A tile adjacent to the blank space can slide into the space
- Objective: reach a specified goal state



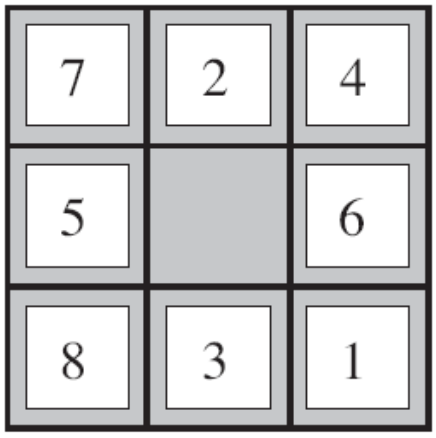
Start State



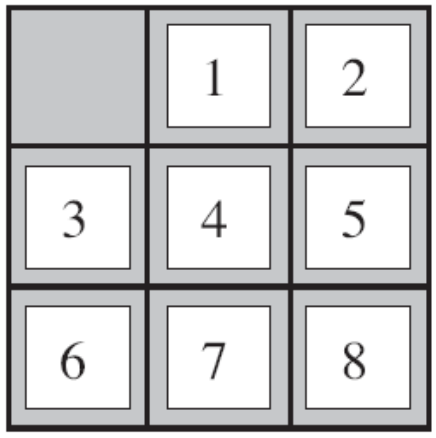
Goal State

The 8-puzzle

- States?
  - any location combination of 8 tiles and the blank
- Initial state?
  - given
- Actions?
  - Movements of the blank space: left, right, up, down, or a subset of these



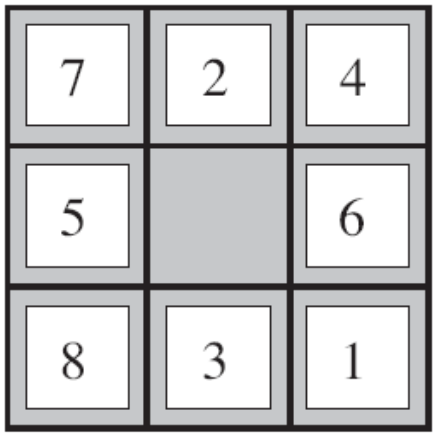
Start State



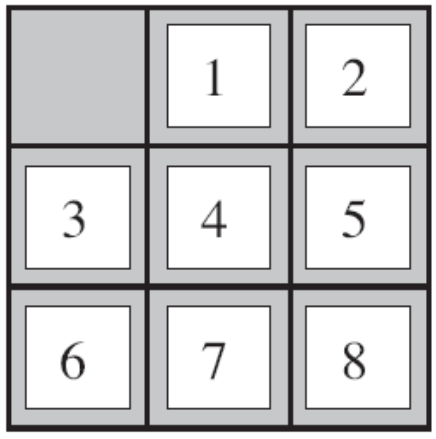
Goal State

The 8-puzzle

- Transition model?
  - given a location combination and a movement, this returns the resulting location combination
- Goal test?
  - check whether the state matches the goal configuration
- Path cost?
  - (each step costs 1) the number of steps in the path



Start State



Goal State

- Donald Knuth (1964): starting with the number 4, a sequence of factorial, square root, and floor operations will reach any desired positive integer.

$$\left\lfloor \sqrt{\sqrt{\sqrt{\sqrt{\sqrt{(4!)!}}}}} \right\rfloor = 5$$

- States?
  - Positive numbers
- Initial state?
  - 4
- Actions?
  - Apply factorial, square root, or floor operation

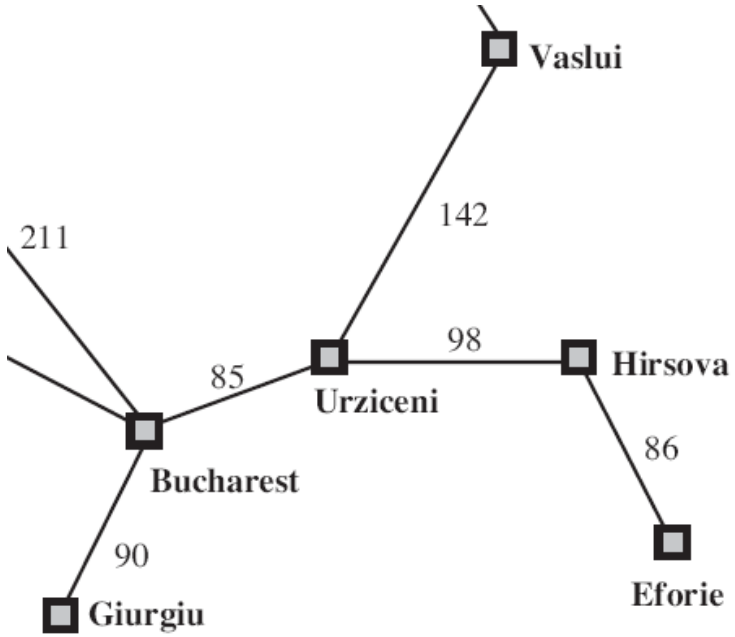
- Donald Knuth (1964): starting with the number 4, a sequence of factorial, square root, and floor operations will reach any desired positive integer.

$$\left\lfloor \sqrt{\sqrt{\sqrt{\sqrt{\sqrt{(4!)!}}}}} \right\rfloor = 5$$

- Transition model?
  - Definition of the operations
- Goal test?
  - State is the desired positive integer
- Infinite state space

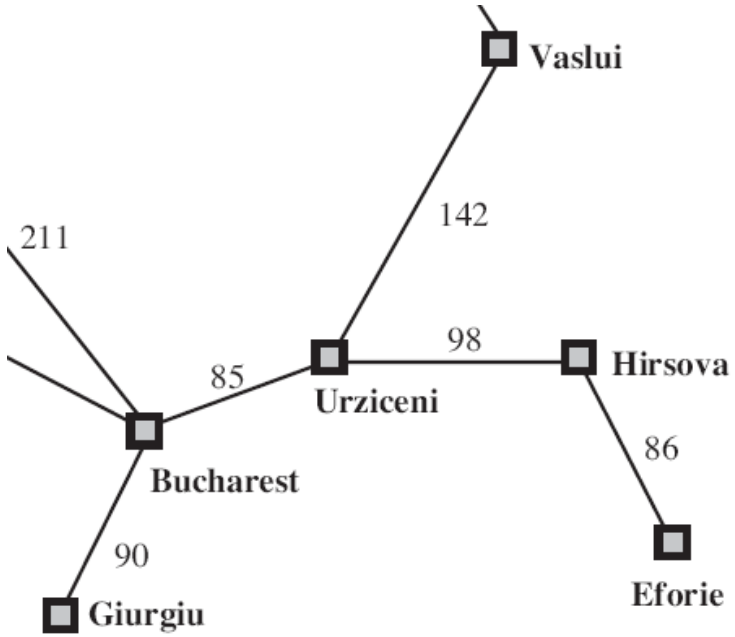
Touring problems

- "Visit every city in Romania at least once, starting and ending in Bucharest"
- Each state?
  - includes the
    current location, and
    the set of cities already visited
- Initial state?
  - In(Bucharest),
    Visited(Bucharest)



Touring problems

- "Visit every city in Romania at least once, starting and ending in Bucharest"
- Intermediate state (typical):
  - In(Vaslui), Visited({Bucharest, Urziceni, Vaslui})
- Goal test:
  - check whether the agent is in Bucharest, and all 20 cities have been visited



River Crossing Problem





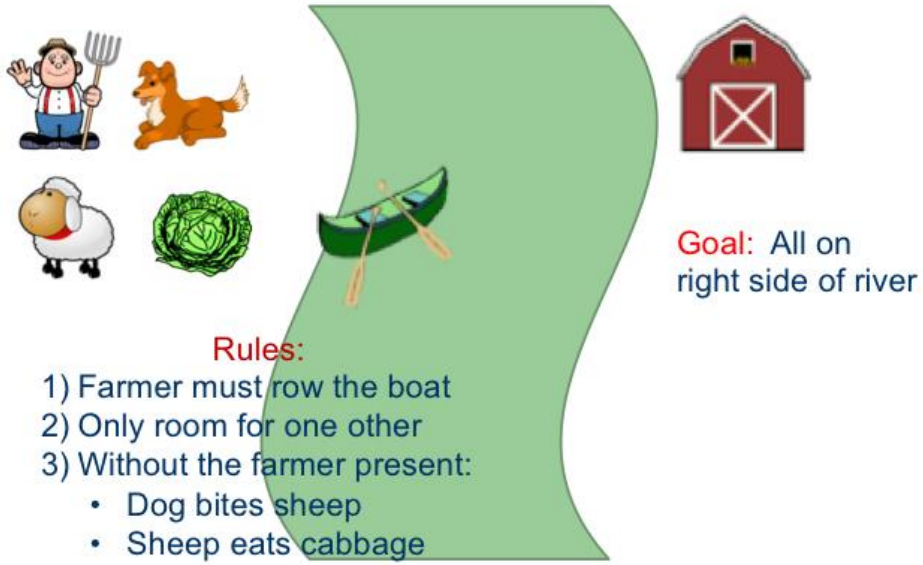

Goal: All on right side of river

Rules:

1) Farmer must row the boat
2) Only room for one other passenger
3) Without the farmer present:
   - Dog bites sheep
   - Sheep eats cabbage

Assume: a **state** describes who is (are) on the right side of the river

River Crossing Problem

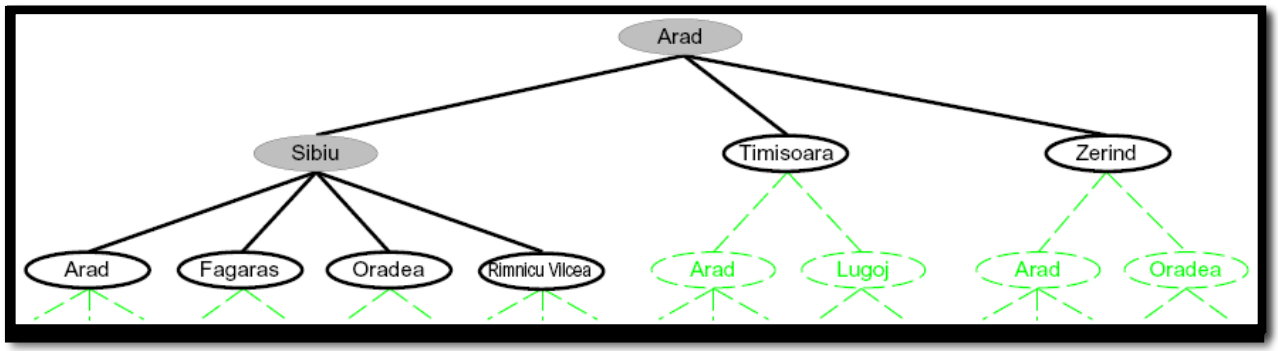


- Cost function?
  - Number of trips

- Goal Test? (Goal state)
  - Everyone is on the right side of the river
- States?
  - D, S, C, FS, DC, FDC, FDS, FCS, FDCS
- Possible Actions?
  - F>, F<, FC>, FC<, FD>, FD<, FS>, FS<

Searching for Solutions

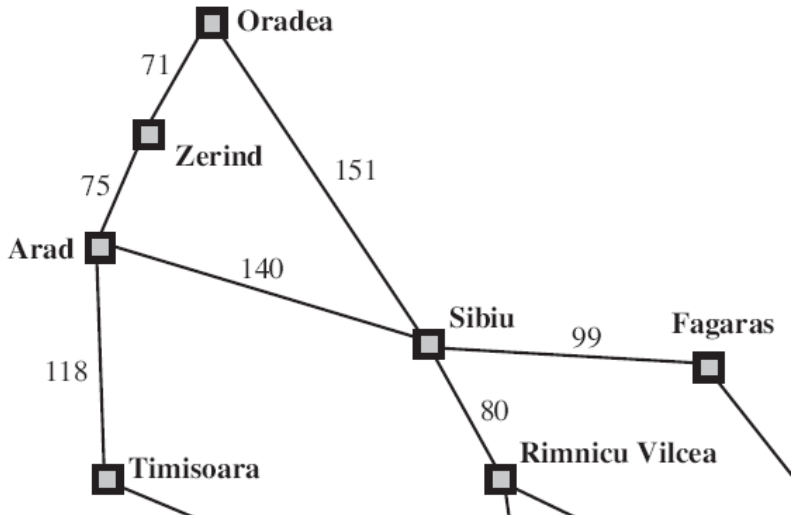
- Having formulated the problems, how do we solve them?
  - Solution: a sequence of actions leading to the goal state
  - Searching algorithms: consider various possible action sequences



- **Search Tree**
  - **Root:** initial state
    - In(Arad)
  - **Branches:** actions
  - **Nodes:** states in the state space
    - Parent node, child nodes
    - Leaf node: a node with no children

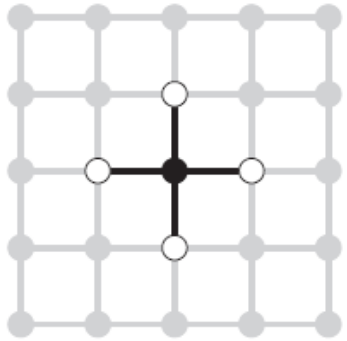
Tree-Search

- We expand the current state and generate a new set of states
- Search Strategy: how to choose which state to expand next?
- Repeated state
  - Arad-Sibiu-Arad
  - Loopy path
    - The complete search tree for Romania is *infinite*
    - Can cause certain algorithms to fail



- Redundant paths
  - More than one way to get from one state to another
    - Arad-Sibiu (140 km long) vs. Arad-Zerind-Oradea-Sibiu (297 km long)

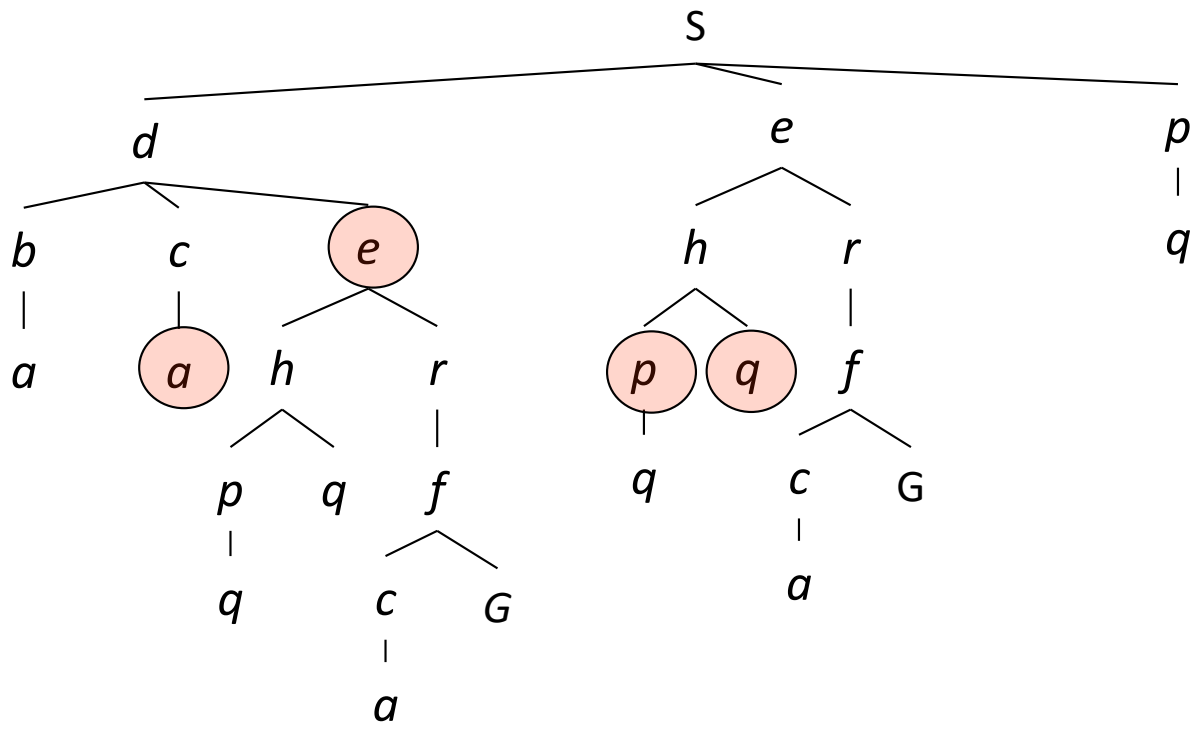
Tree-Search



- Route-finding on a rectangular grid (suppose it is infinitely large)
  - Each state: 4 successors
  - A search tree of depth d that includes repeated states: 4^d leaves
  - Only about $2d^2$ distinct states within d steps of any given state
    - $d = 20$, a trillion nodes, but only about 800 distinct states
- Algorithms that forget their history are doomed to repeat it!
  - Avoid exploring redundant paths by remembering where one has been
    - Graph-Search

Graph-Search

- Idea: never expand a state twice
- Example: in the following breadth-first graph search, we do not expand the circled nodes

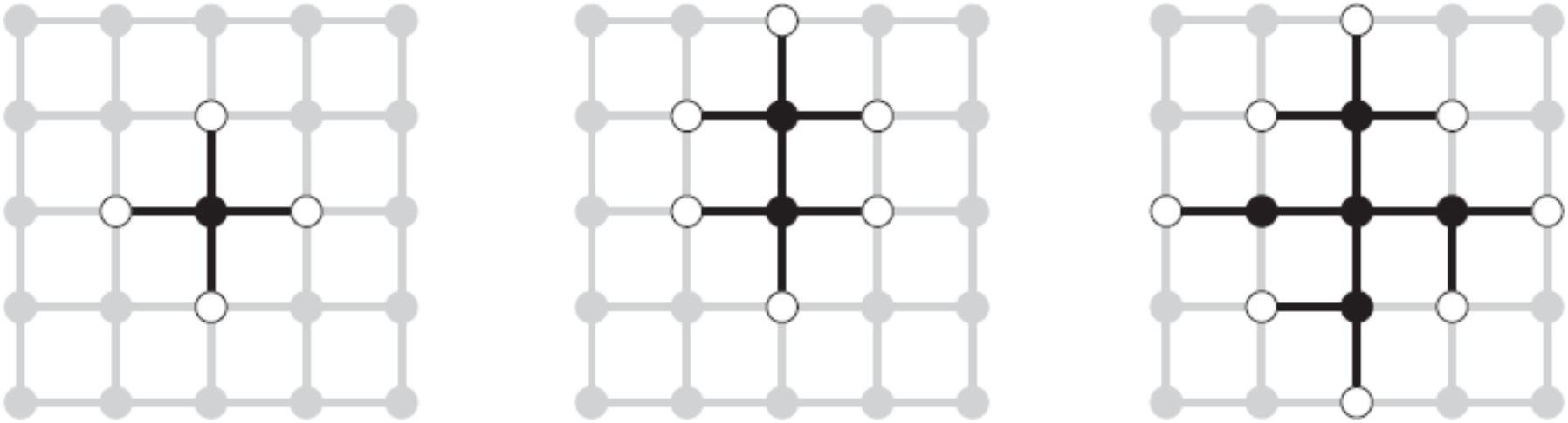


Graph-Search

- Idea: never expand a state twice
- How to implement:
  - Tree search + set of expanded states ("closed set")
  - Expand the search tree node-by-node, but…
  - Before expanding a node, check to make sure its state has never been expanded before
  - If not new, skip it, if new add to closed set
- Important: store the closed set as a set, which does not have repeated elements

Graph-Search

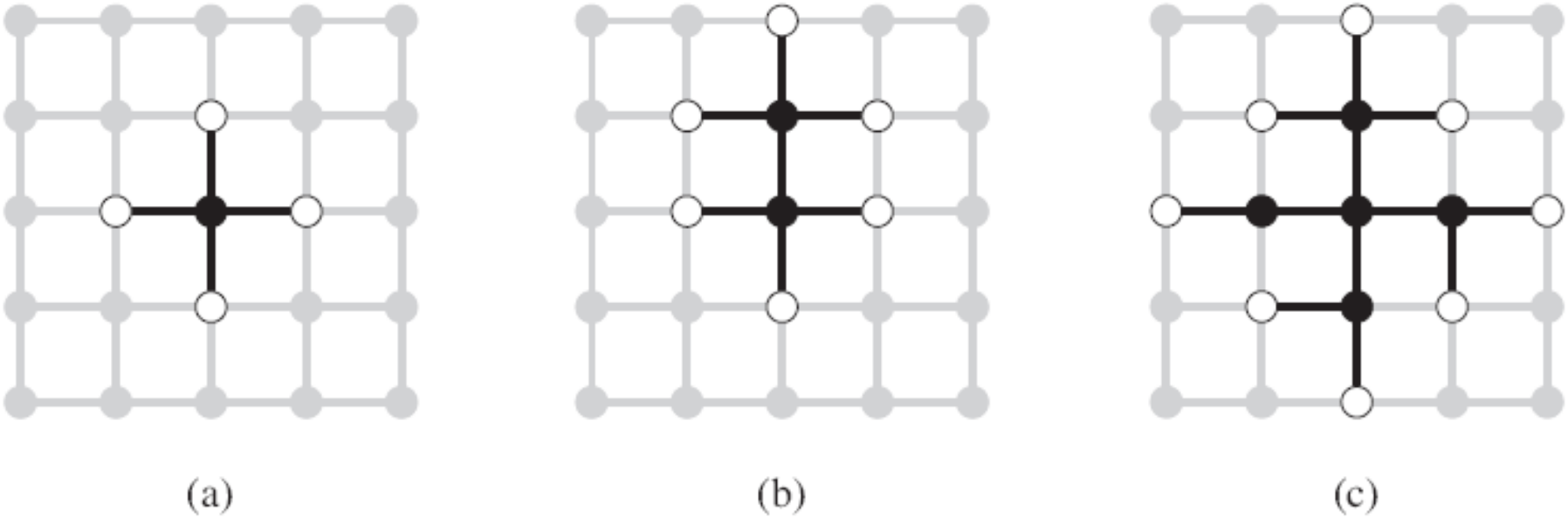
- Rectangular-grid problem
  - Black nodes: the explored region of the state space
  - White nodes: frontier (to be explored)
  - Gray nodes: unexplored region



(a) (b) (c)

Graph-Search

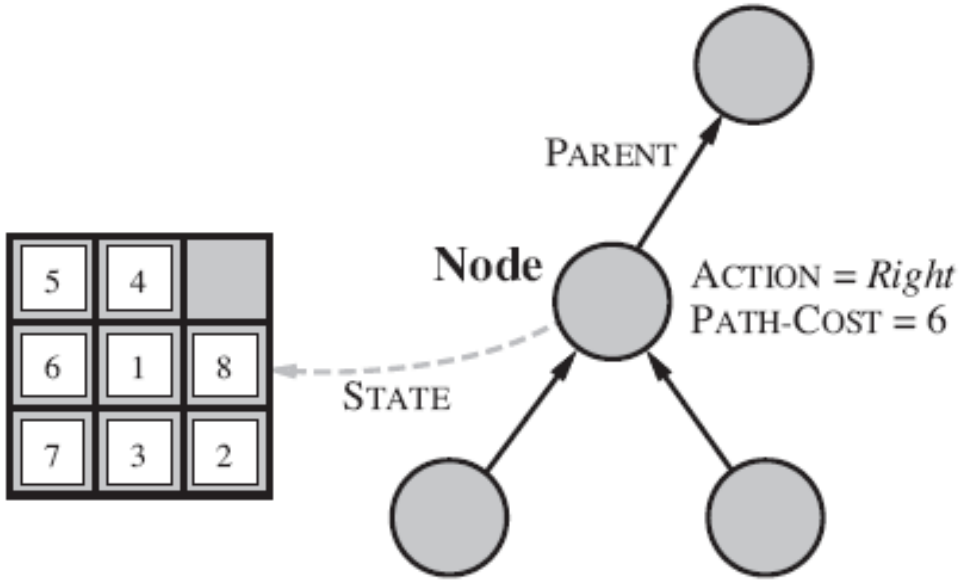
- Separation Property
  - The frontier (white) always separates the explored (black) from the unexplored (gray)
  - (a) just the root has been explored
  - (b) one leaf node has been expanded
  - (c) the remaining successors of the root have been expanded in clockwise order



(a) (b) (c)

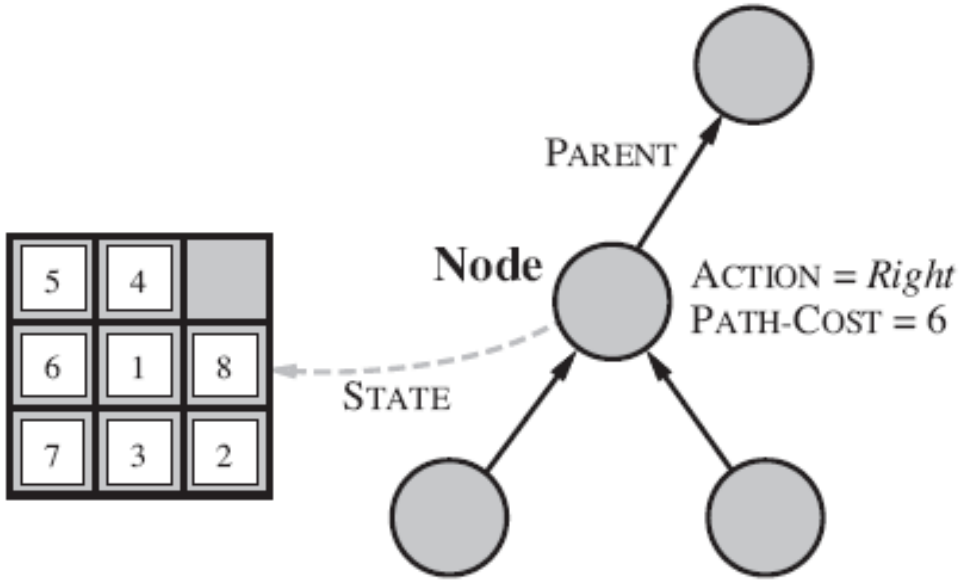
Data Structure of Search Algorithms

- Data structure: keeps track of the search tree
- Data structure for node n
  - n.State: the state in the state space to which the node corresponds.
  - n.Parent: the node in the search tree that generated this node.
  - Arrows point from child to parent



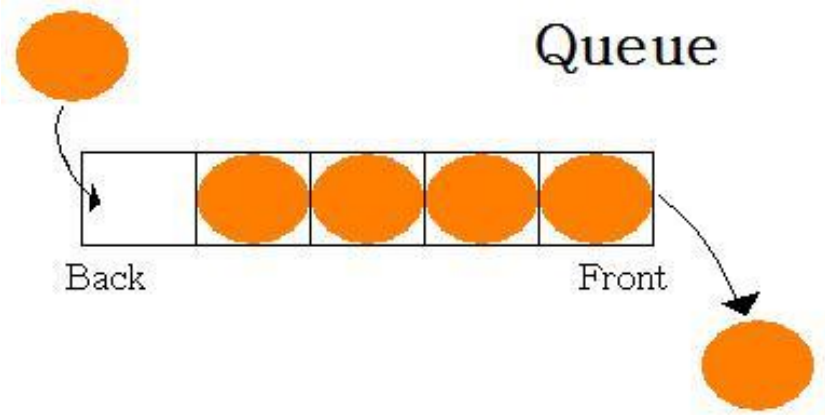
Data Structure of a Node

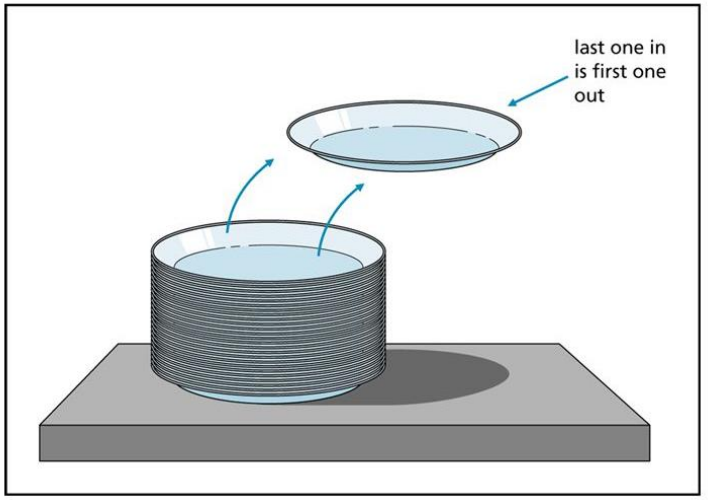
- Data structure for node n
  - n.Action: the action that was applied to the parent to generate the node.
    - n.Action: movement of the blank tile
    - Draw the state of the parent node?
  - n.Path-Cost: the cost, denoted by $g(n)$, of the path from the initial state to the node, as indicated by the parent pointers.



Frontier Set

- The set of Frontier nodes
  - a FIFO queue, or
  - a LIFO stack, or
  - a Priority queue





Performance Measurement

- Completeness: Is the algorithm guaranteed to find a solution when there is one?

- Optimality: Does the strategy find the optimal solution (w.r.t some performance measure)?
- Time complexity: How long does it take to find a solution?
  - Measured in terms of the number of nodes generated/explored/expanded/visited during the search

- Space complexity: How much memory is needed to perform the search?
  - Measured in terms of the maximum number of nodes stored in memory

Parameters

- b - Branching factor
  - Maximum number of successors of any node
- d - The *depth* of the shallowest goal node (shallowest solution)
  - i.e. the number of steps along the path from the root
- m – the *maximum length* of any path in the state space
  - For tree search
    - m can be much larger than d
    - m is infinite if the tree is unbounded

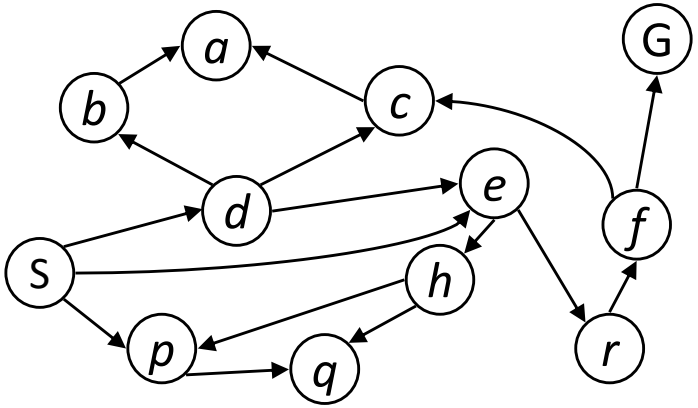
Performance Measurement

- Search Cost
  - Time complexity: the amount of time taken by the search
- Path Cost
  - The path cost of the solution found (e.g. the total length of the path in kilometers)

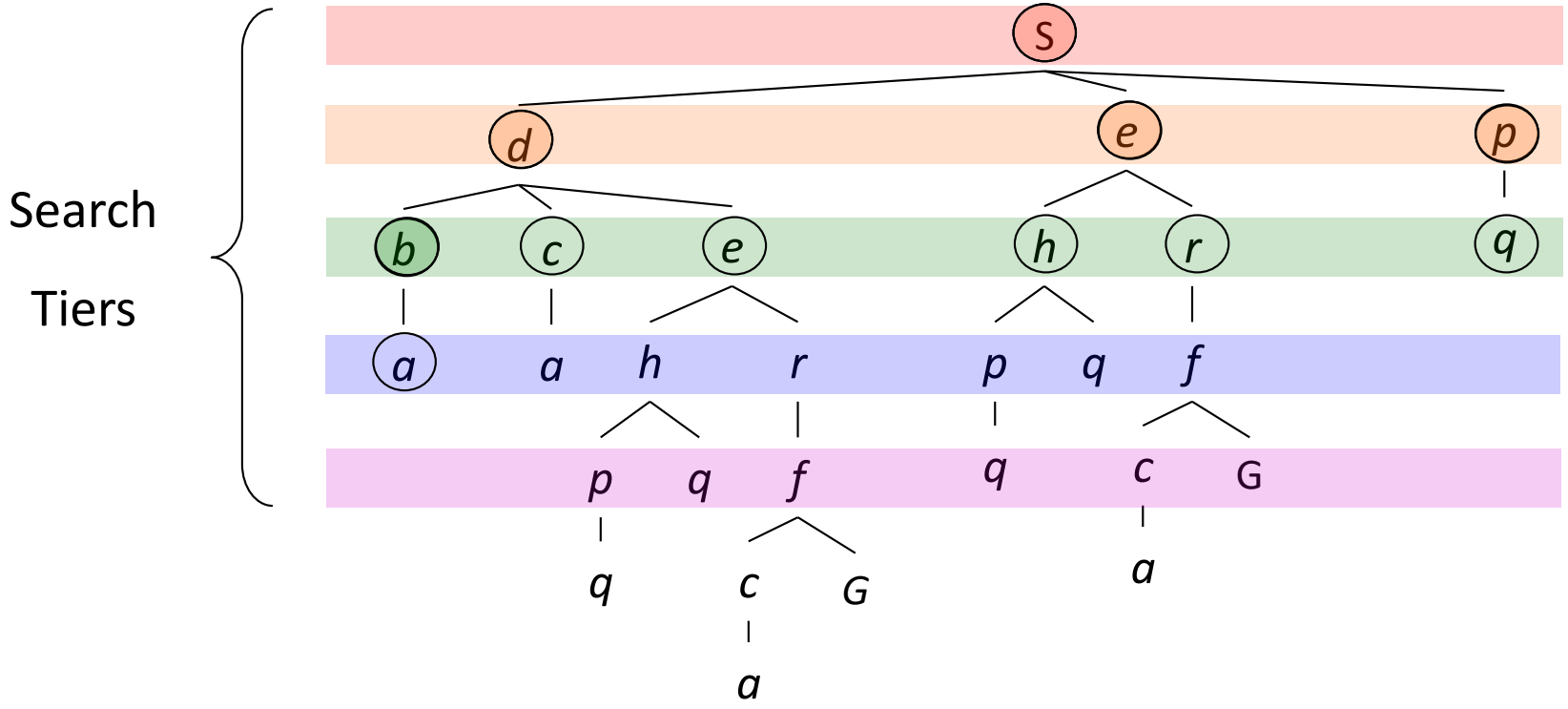
Uninformed Search

- Also called: Blind Search
- The strategies have no additional information about states
  - e.g. no information about how close the states are to the goal state
- Can only
  - Generate successors
  - Distinguish a goal state from a non-goal state
- You have no clue whether one non-goal state is better than any other
- There are different uninformed search strategies
  - They are distinguished by the **order** in which nodes are expanded

Breadth-First Search

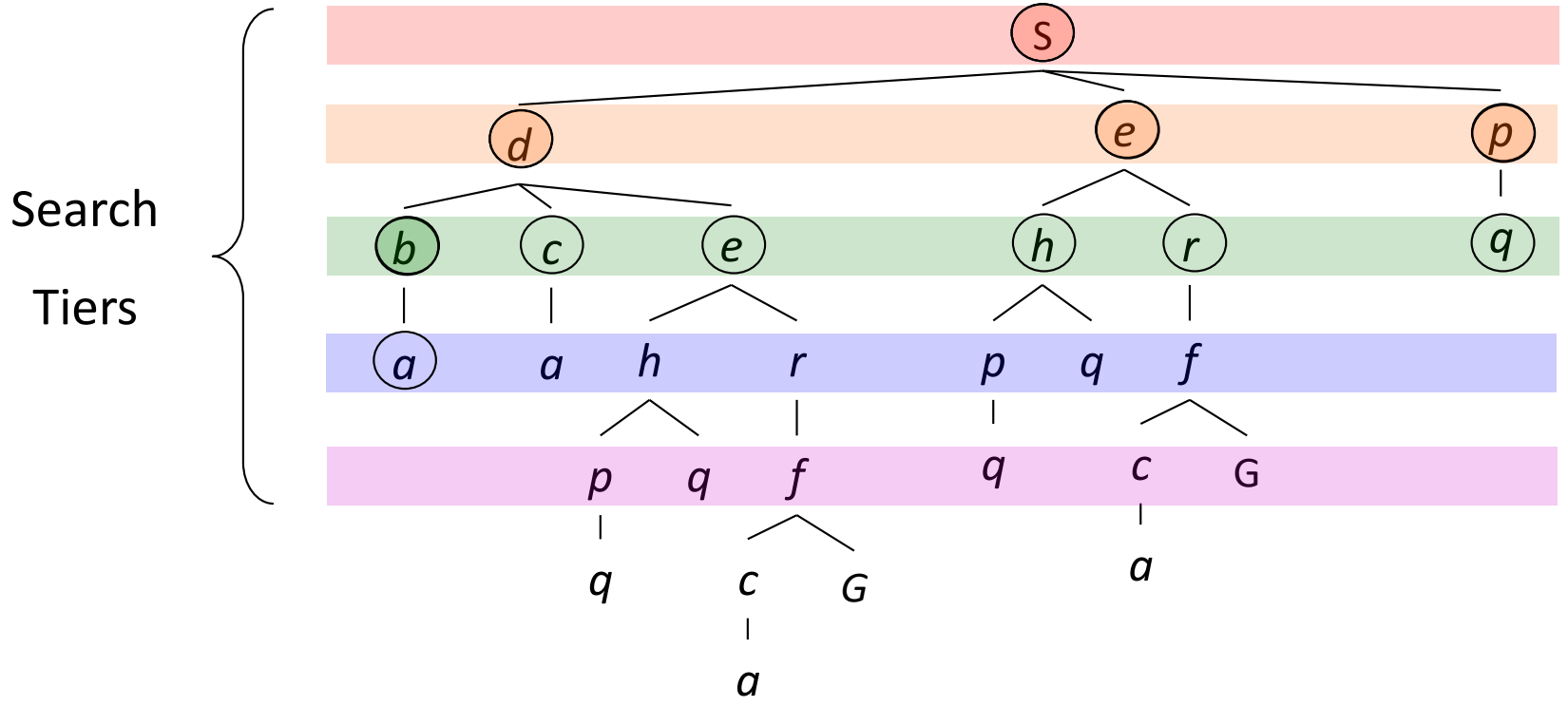


Initial State: S
Goal State: G
Explored: color circled
Frontier: white circled
Unexplored: uncircled



Breadth-First Search

- All the nodes are expanded at a given depth in the search tree before any nodes at the next level are expanded
- The shallowest unexpanded node is chosen for expansion
  - Achieved by using a FIFO queue for the frontier

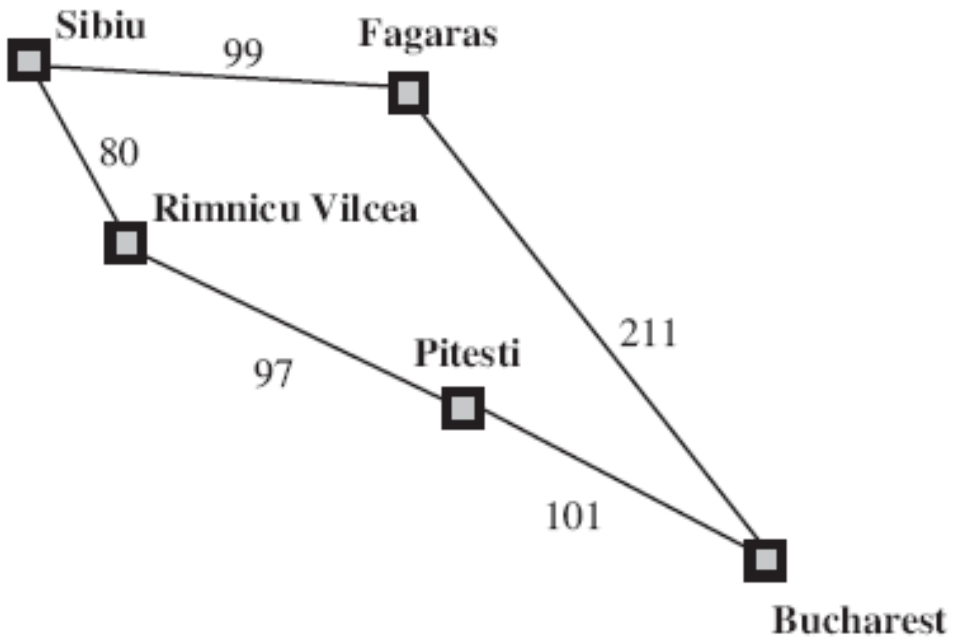


Breadth-First Search

- Optimality
  - In terms of solution path cost
  - Optimal when all step costs are equal (i.e. deeper solutions are always less optimal)
  - Always expands the shallowest unexpanded node
- Complete?
  - Yes if the branching factor b is finite

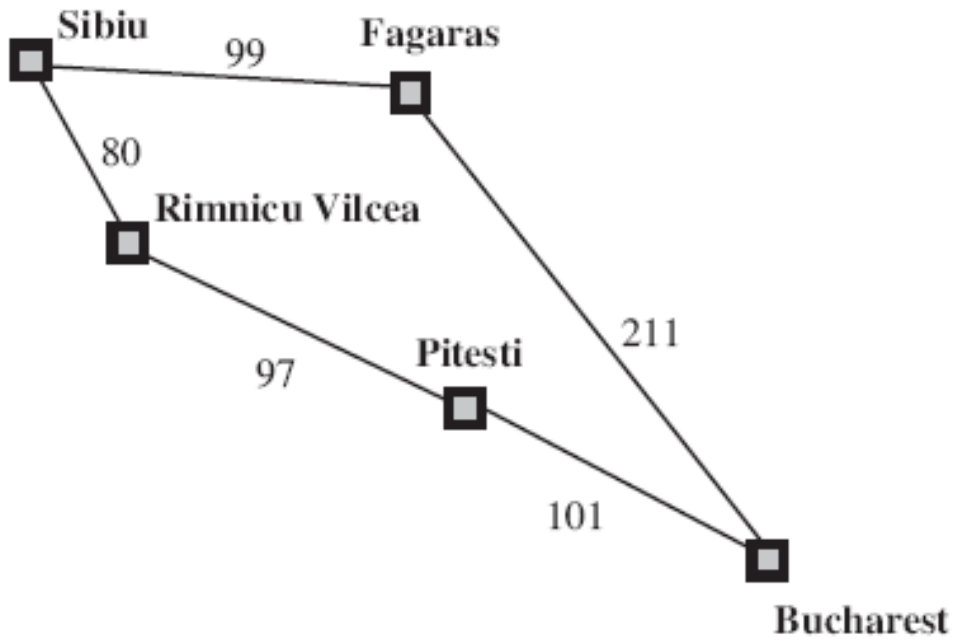
Uniform-cost Search

- Expands the node n with the *lowest path cost* $g(n)$
  - Done by storing the frontier as a priority queue ordered by g
  - *Path cost* $g(n)$: the cost of the path from the initial state to the node
- Example: get from Sibiu to Bucharest



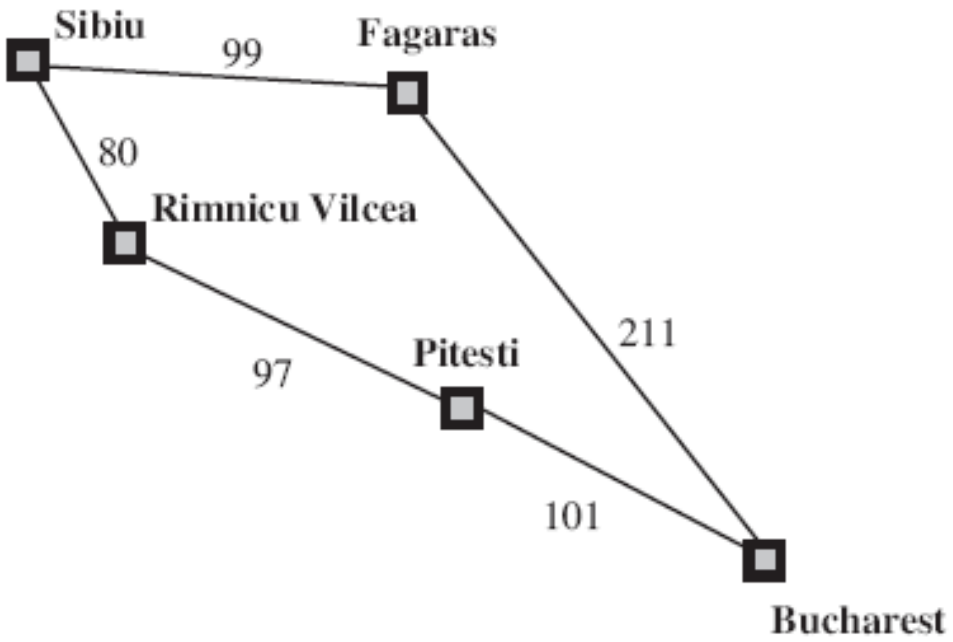
Uniform-cost Search

- Guided by path costs (rather than steps)
- Does not care about the number of steps a path has
- Only cares about the total cost of a path



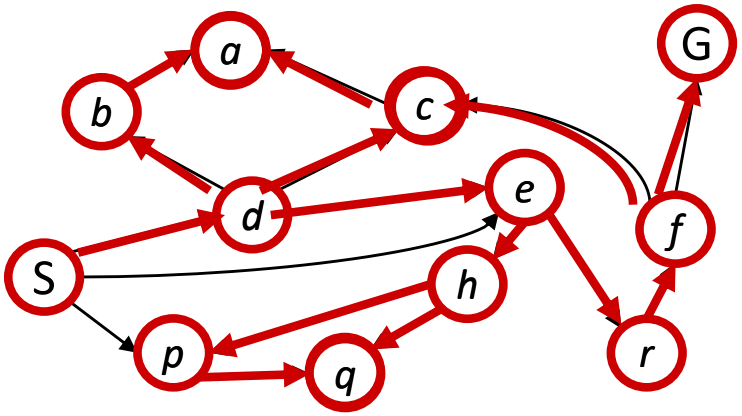
Uniform-cost Search

- Optimal or not?
  - Yes



- Completeness?
  - Guaranteed if the cost of every step is greater than a small positive value ε
  - If there's a path with an infinite sequence of zero-cost actions, then it will get stuck in an infinite loop

Depth-First Search



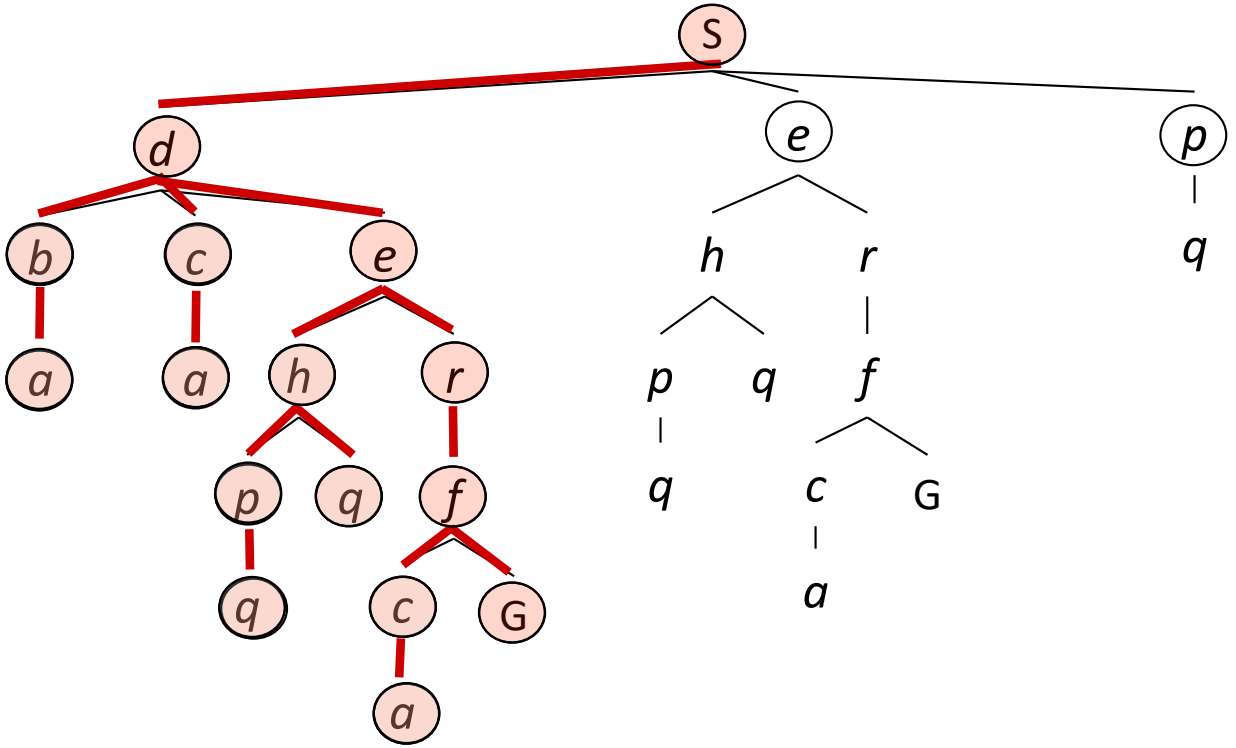
Initial State: S

Goal State: G

Explored: color circled

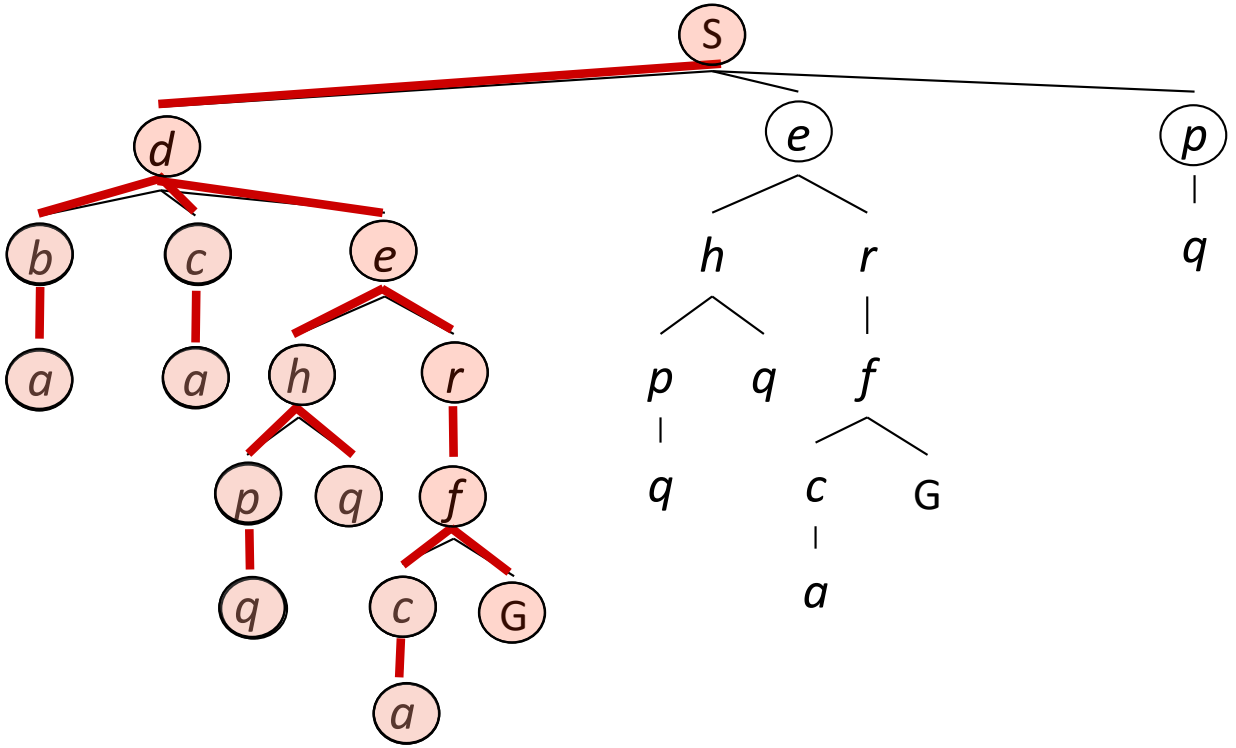
Frontier: white circled

Unexplored: uncircled



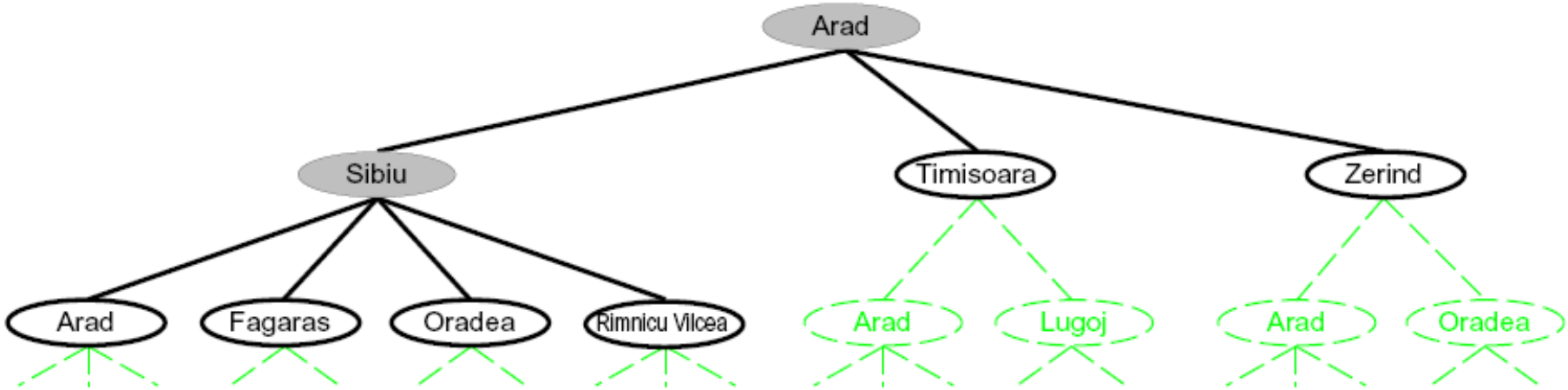
Depth-First Search

- Expands the deepest node in the current frontier of the search tree
- LIFO stack – the most recent generated node is chosen for expansion
- Explored nodes with no descendants are removed from memory



Depth-First Search

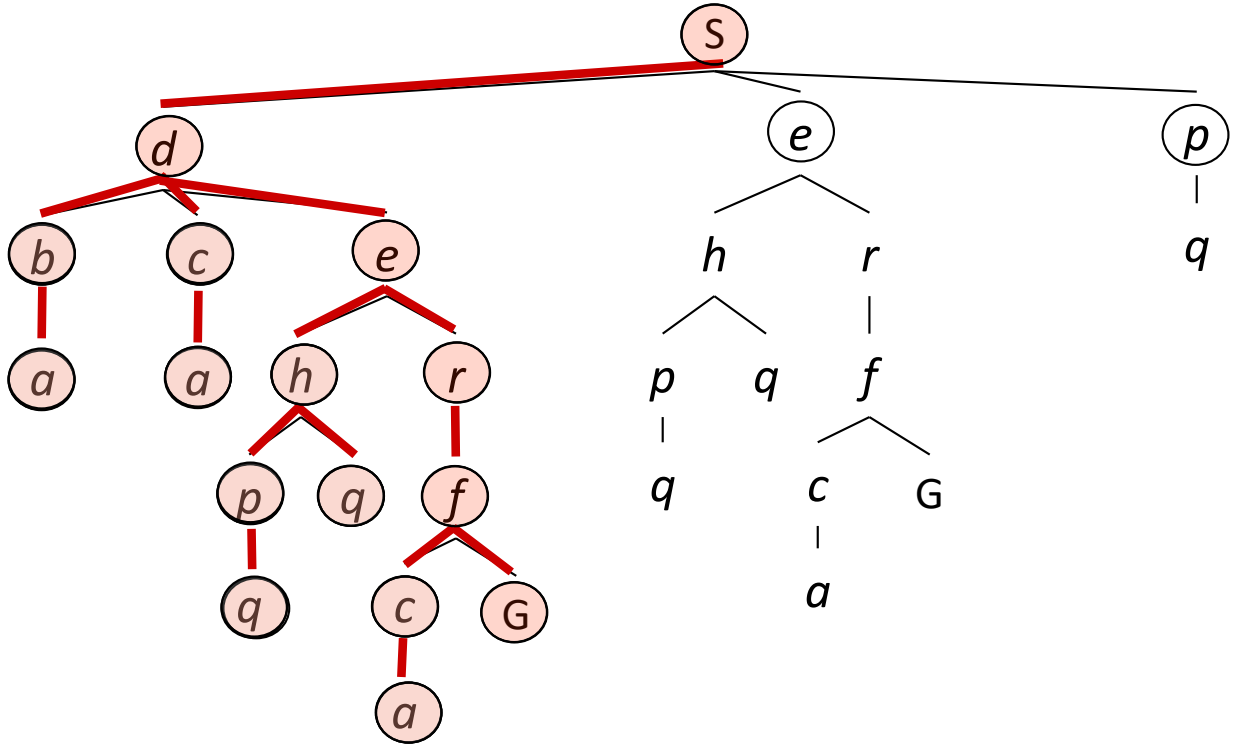
- Complete or not?
  - Depth-first Graph-search: Yes (avoids repeated states)
  - Depth-first Tree-search: No



- Arad-Sibiu-Arad-Sibiu loop forever!

Depth-First Search

- Optimal or not? (in terms of the cost of the solution path)
- (hint: If node **e** is the goal node …)



- Answer: No!
- The solution returned by the DFS will get to it in 2 steps instead of 1 step

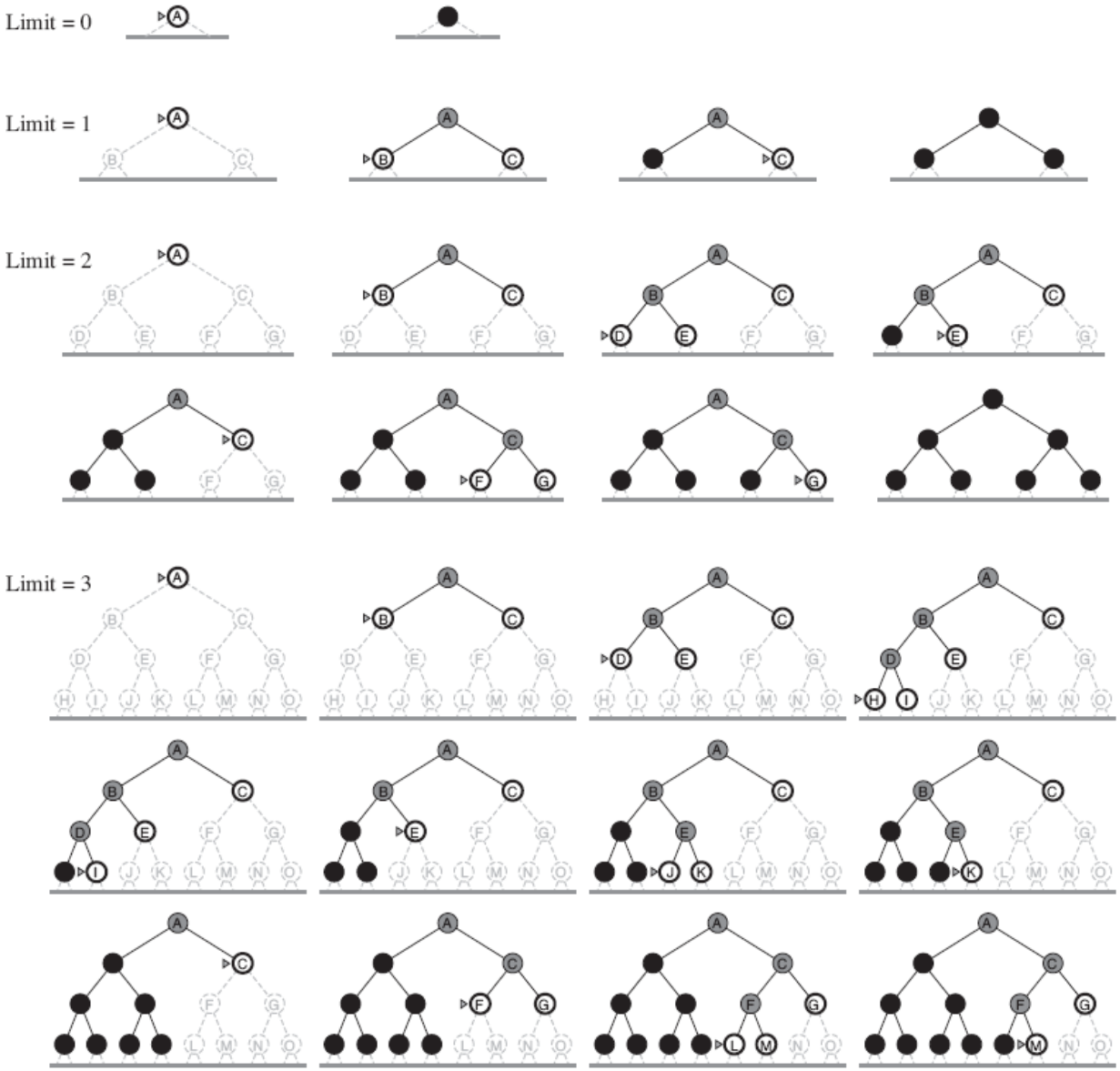
Depth-Limited Search

- Supply DFS with a predetermined depth limit l
  - Solves the infinite path problem
- Complete or not?
  - If $l < d$: incomplete! (e.g. when d is unknown)
- Optimal or not?
  - If $l > d$: Not guaranteed.
- Two kinds of failure
  - *Standard* failure: no solution
  - *Cutoff* failure: no solution within the depth limit

Iterative Deepening DFS

- Repeatedly applies depth-limited search with increasing limits l
- Terminates an iteration when a solution is found or if the depth-limited search returns *failure* (no solution for that depth limit)
- Combines the benefits of DFS and BFS

Iterative Deepening DFS

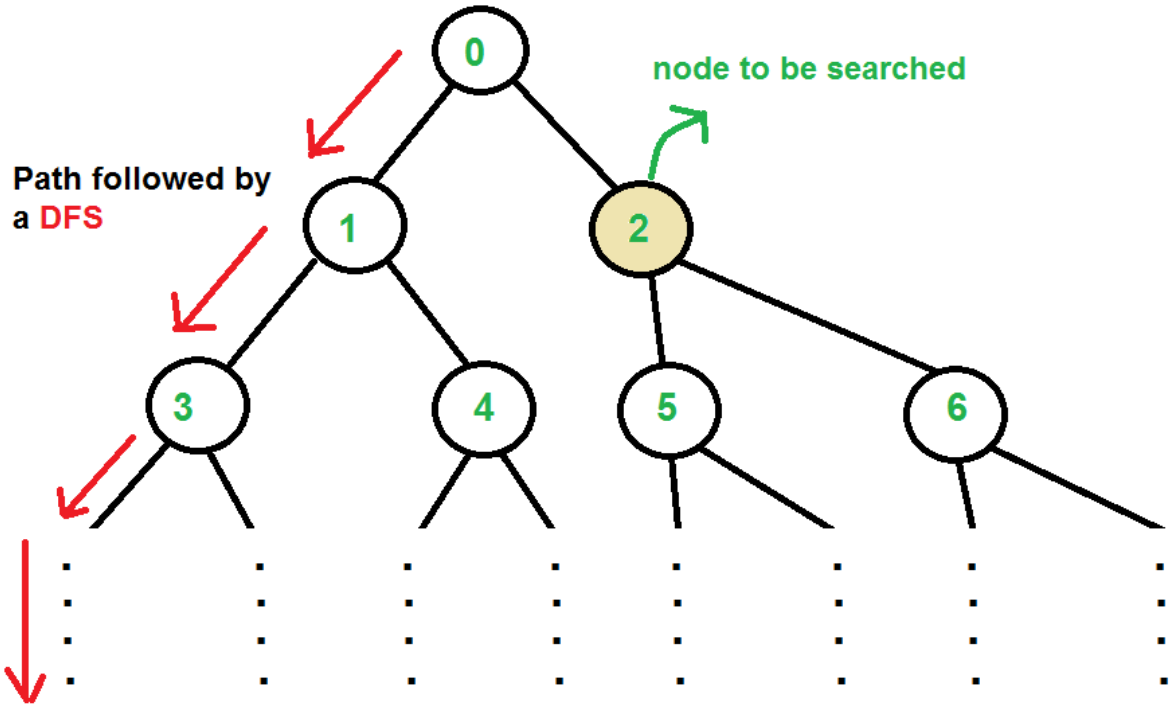


Four iterations of iterative deepening search on a binary tree

Suppose: M is the goal node

Black circles: nodes removed from memory

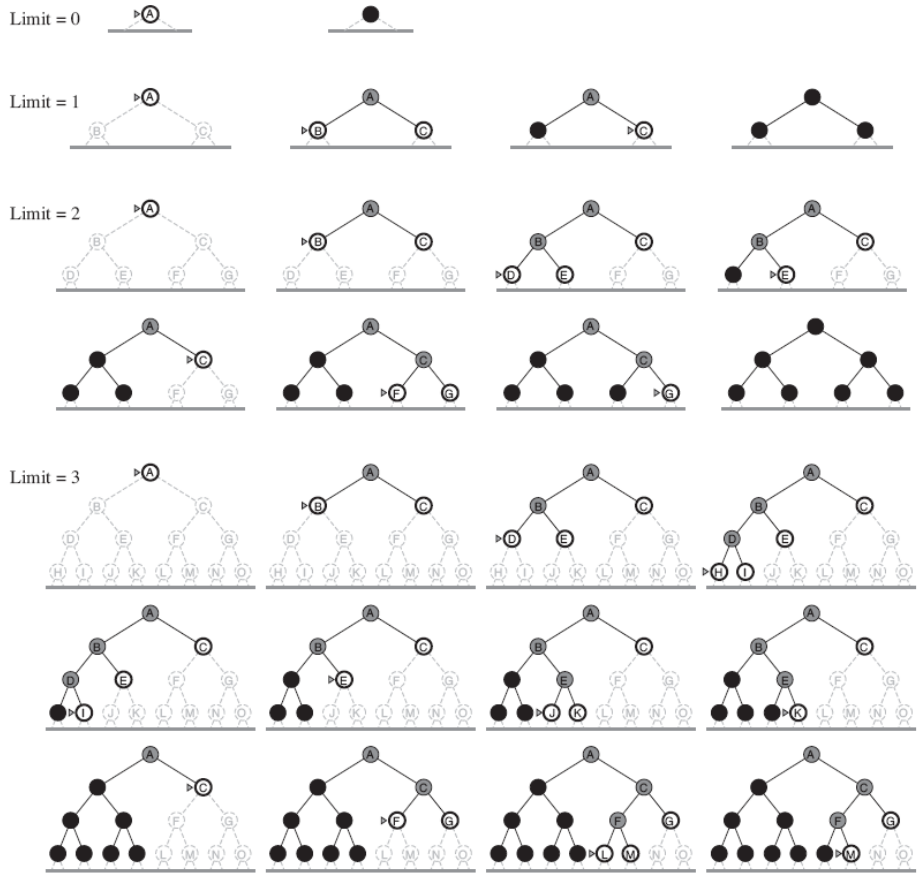
Iterative Deepening DFS



- We want to find node ‘2’ of the given “deep” tree.
- A DFS starting from node ‘0’ will dive left, towards node 1 and so on
- Hence, a DFS wastes a lot of time in coming back to node 2
- An Iterative Deepening DFS overcomes this and quickly finds the desired node.

Iterative Deepening DFS

- Complete? Yes when b is finite
- Optimal? Yes in the sense that it can always find the shallowest solution.



Informed Search Strategies

- Uses problem-specific knowledge beyond the definition of the problem itself
- Can find solutions more efficiently than can an uninformed strategy

Heuristic Function $h(n)$

- Heuristic function

  $h(n)$= estimated cost of the cheapest path from node n to a goal state

- Let $h(n)$ be
  - Arbitrary
  - Nonnegative
  - Problem-specific functions
  - Define: if n is a goal node, then $h(n) = 0$

Greedy Search

- Expands the node that seems closest to the goal
- Evaluates nodes by using a heuristic function $h(n)$
- Example: route-finding problem in Romania
  - Use straight-line distance heuristic h_{SLD}

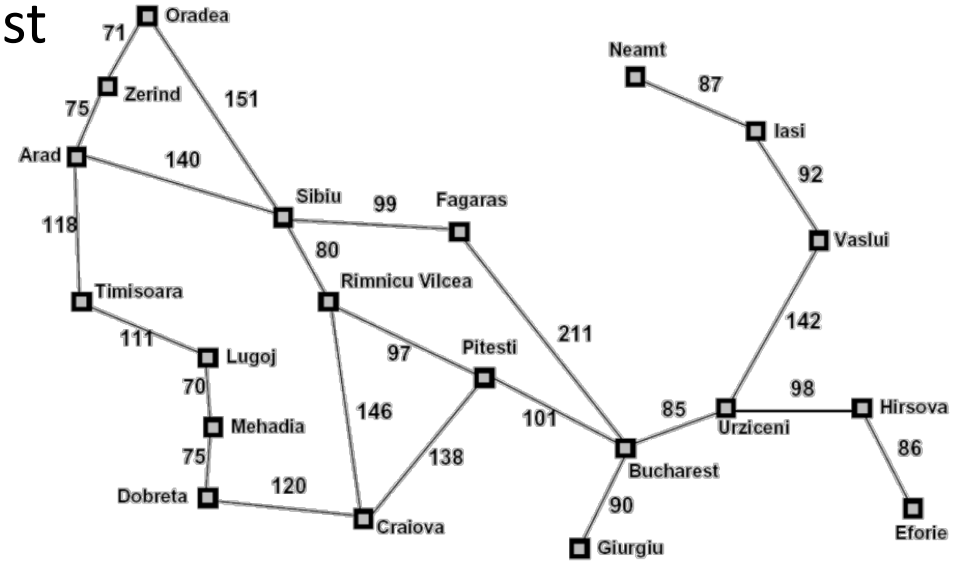
The straight-line distance to Bucharest

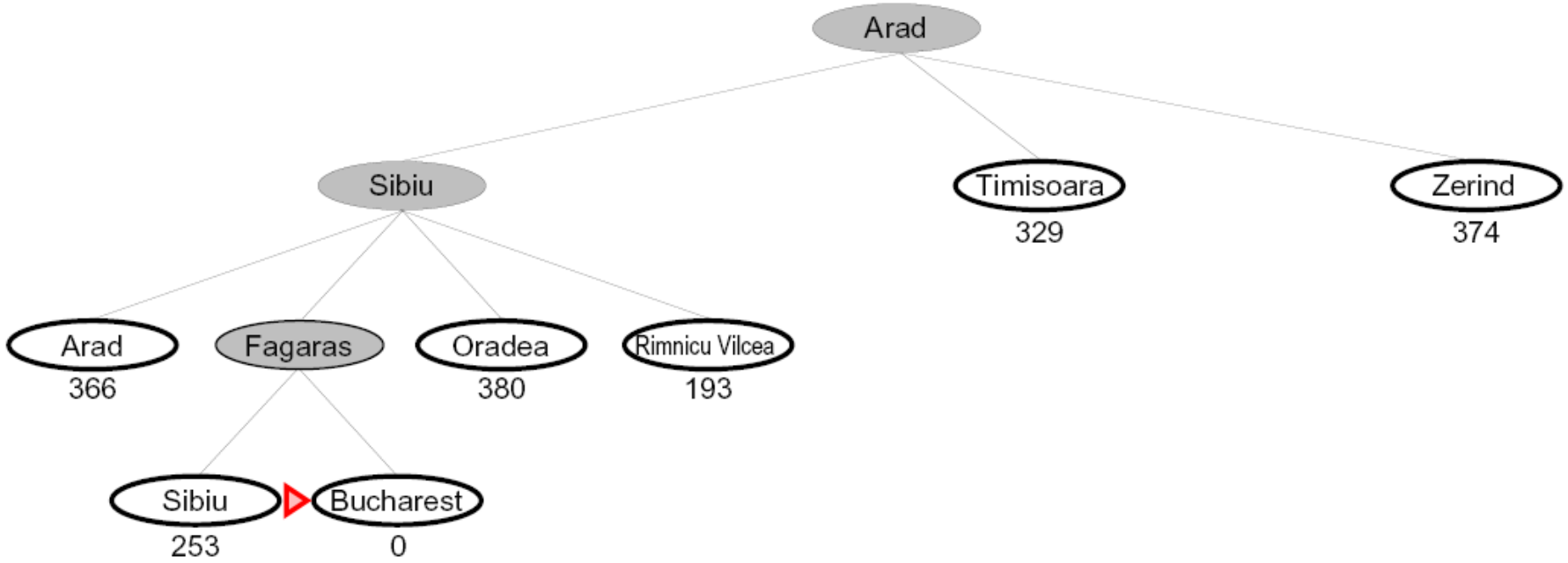
Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Drobeta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374

Greedy Search

The straight-line distance to Bucharest

Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Drobeta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374





Greedy Search

- Why is it called “greedy”?
  - At each step, it tries to get as close to the goal as it can

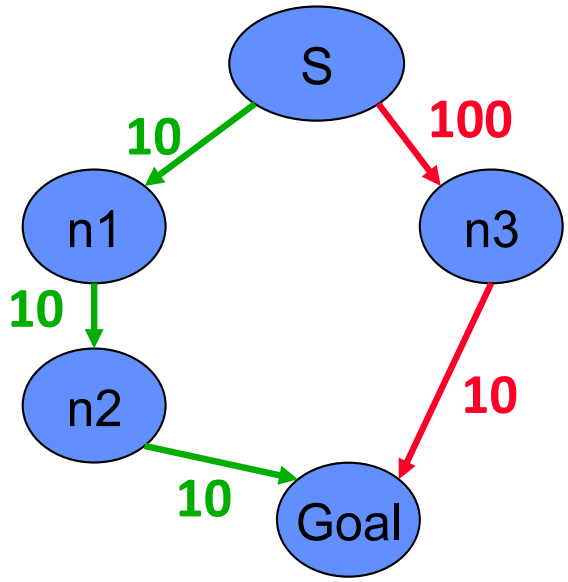
Greedy Search

- Not optimal
  - It ignores the cost of getting to n
  - Can be led astray exploring nodes that cost a lot but seem to be close to the goal

S to n1: → step cost = 10

S to n3: → step cost = 100

h(n1) = 20



h(n3) = 10

Greedy Search

- Completeness
  - Greedy tree search: Incomplete even in a finite state space (does not guarantee to find a solution)
  - The graph search version: Complete in finite state spaces, but not in infinite ones

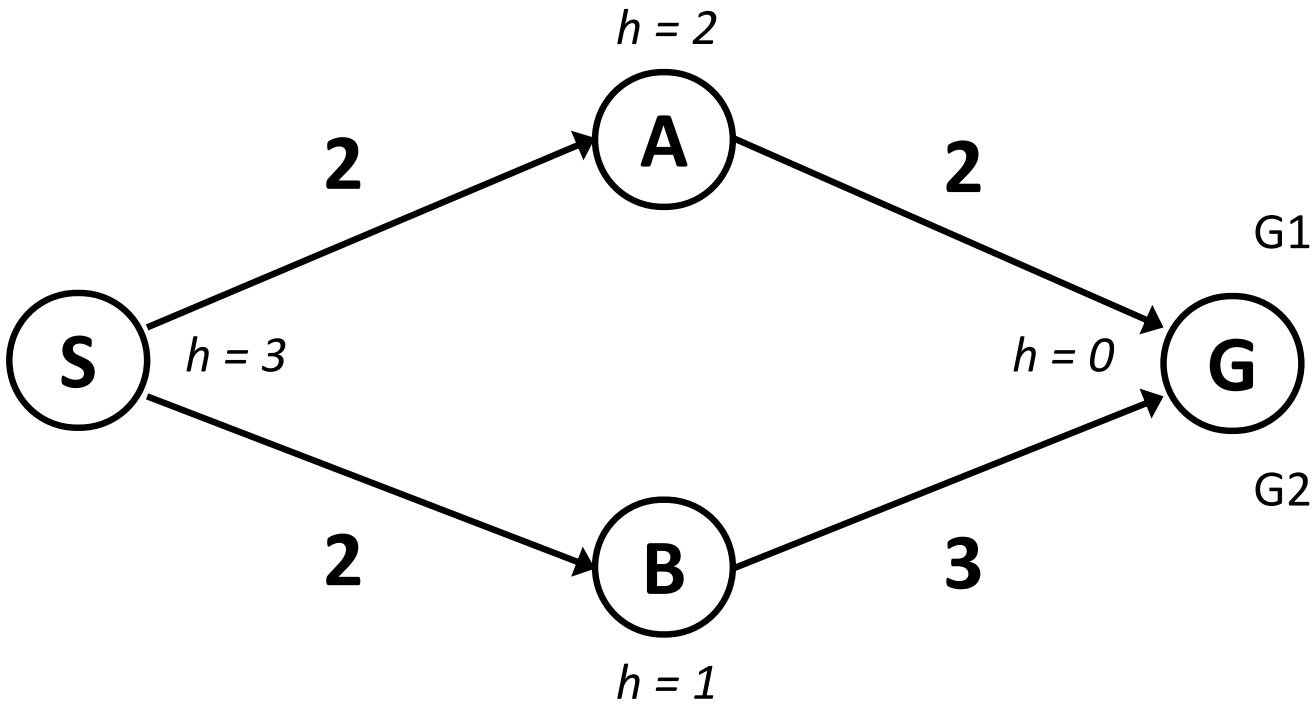
A^* Search

- Minimizes the total estimated solution cost

$$f(n) = g(n) + h(n)$$

  - $g(n)$ - the path cost from the initial node to node n
  - $h(n)$ - the estimated cost to get from node n to the goal node
  - $f(n)$ – estimated cost of the best path that continuous from node n to a goal

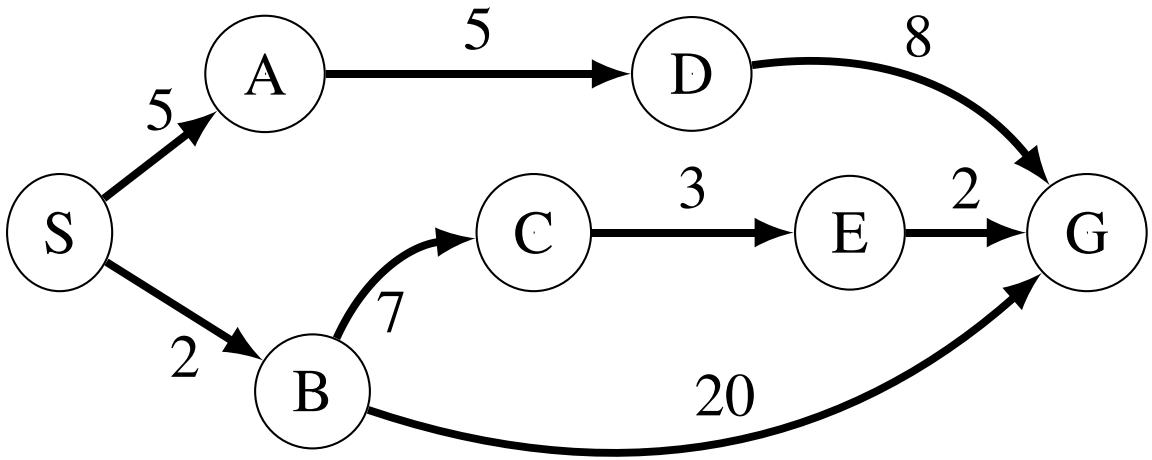
A* Search Example



- Start from S, G is the goal (G=G1=G2, achieved from different path)
- Expanded nodes in order: S, B, A, G1
- Solution path: S->A->G1
- Solution path cost: 4

BFS Example

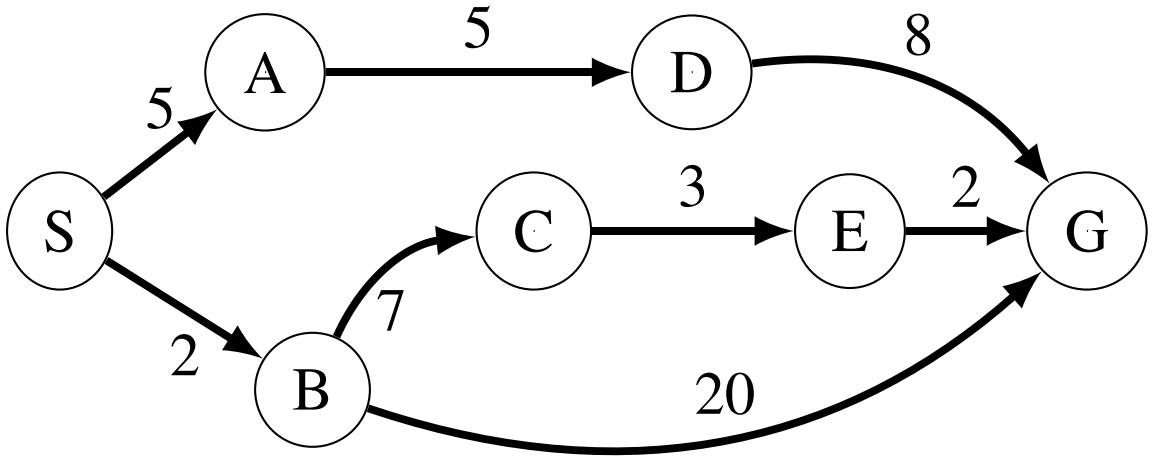
- S is the start node, G is the goal node, use BFS to find a path from S to G. List the expanded nodes in order, give the solution path, and solution path cost. Use alphabetical order to break ties.



- Answer:
- Expanded nodes: S,A,B,D,C,G
- Solution path: SBG
- Path cost: 22

DFS Example

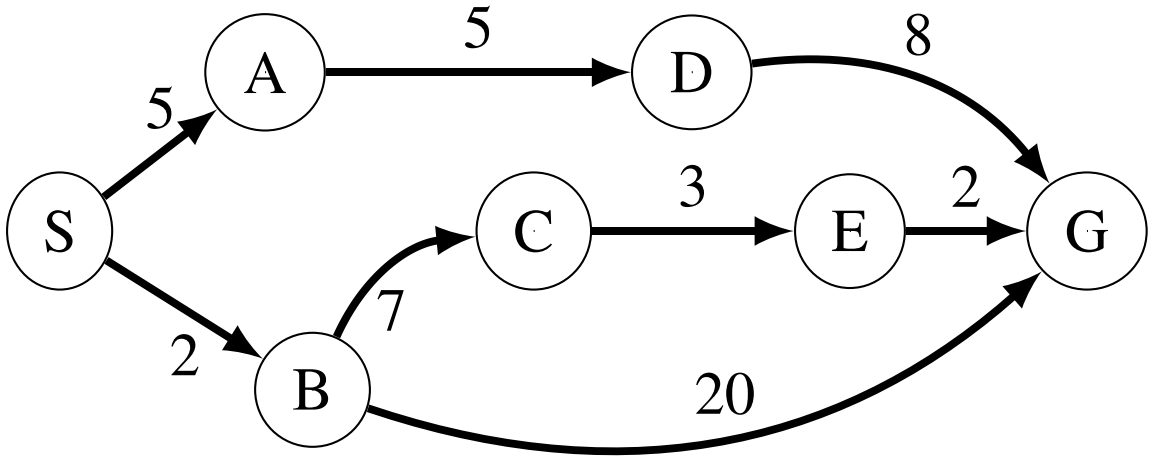
- Solve the same problem by depth-first search.



- Answer:
- Expanded nodes: S,A,D,G
- Solution path: S,A,D,G
- Path cost: 18

IDDFS Example

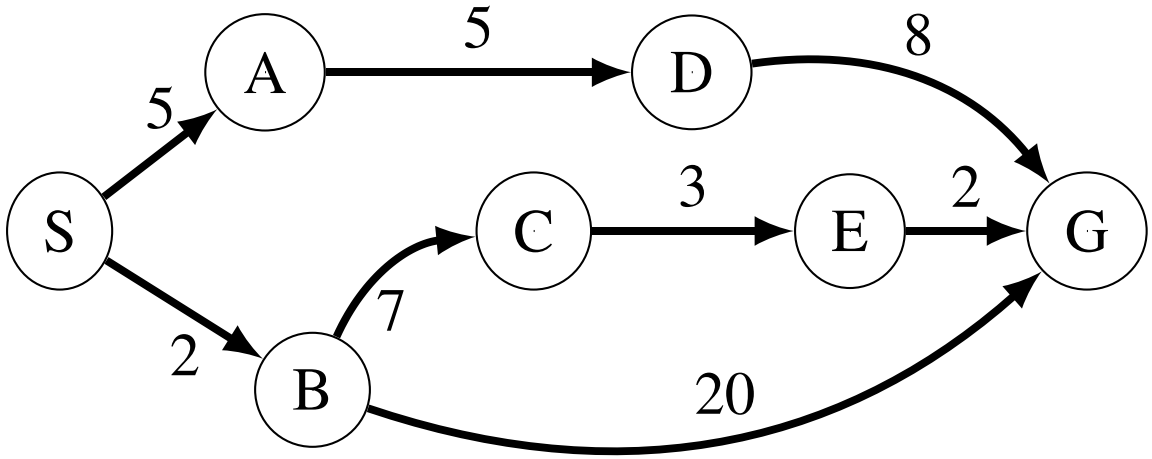
- Solve the same problem by the Iterative Deepening DFS strategy.



- Answer:
- Expanded nodes: S; SAB; SADBCG.
- Solution path: SBG
- Path cost: 22

UCS Example

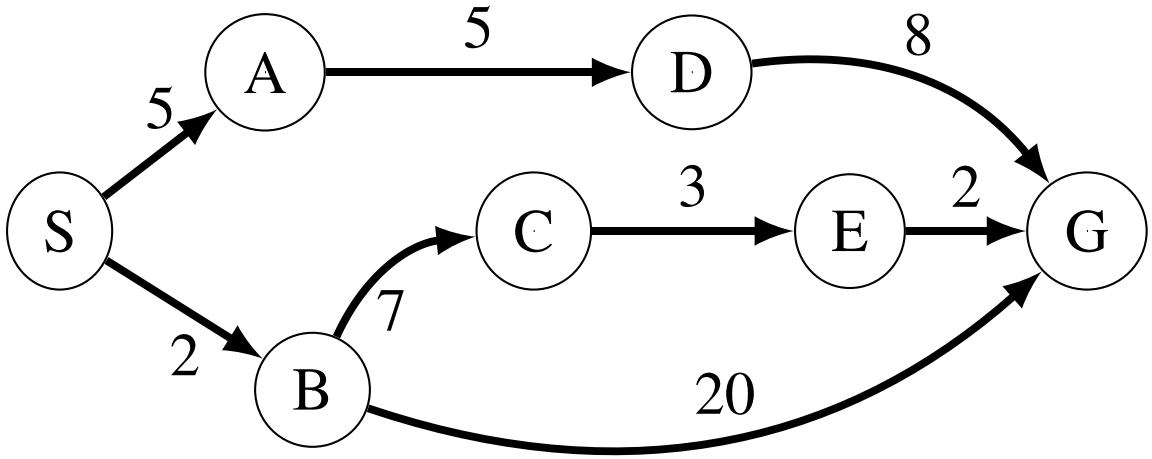
- Solve the same problem by the uniform-cost search.



- Answer:
- Expanded nodes: S,B,A,C,D,E,G
- Solution path: SBCEG
- Path cost: 14

Greedy Search Example

- Solve the same problem by greedy search. The heuristic function values are given.

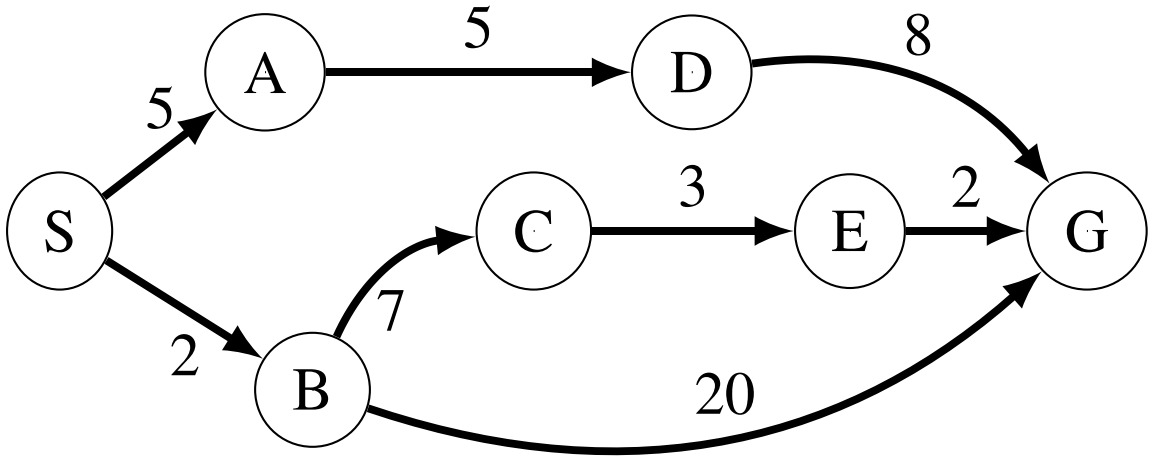


- Answer:
- Expanded nodes: S,A,D,G
- Solution path: S,A,D,G
- Path cost: 18

h(S)=12, h(A)=8,
h(B)=12, h(C)=3,
h(D)=7, h(E)=1,
h(G)=0.

A* Search Example

- Solve the same problem by the A* search strategy.



- Answer:
- Expanded nodes: S,A,B,C,E,G
- Solution path: SBCEG
- Path cost: 14

h(S)=12, h(A)=8,
h(B)=12, h(C)=3,
h(D)=7, h(E)=1,
h(G)=0.