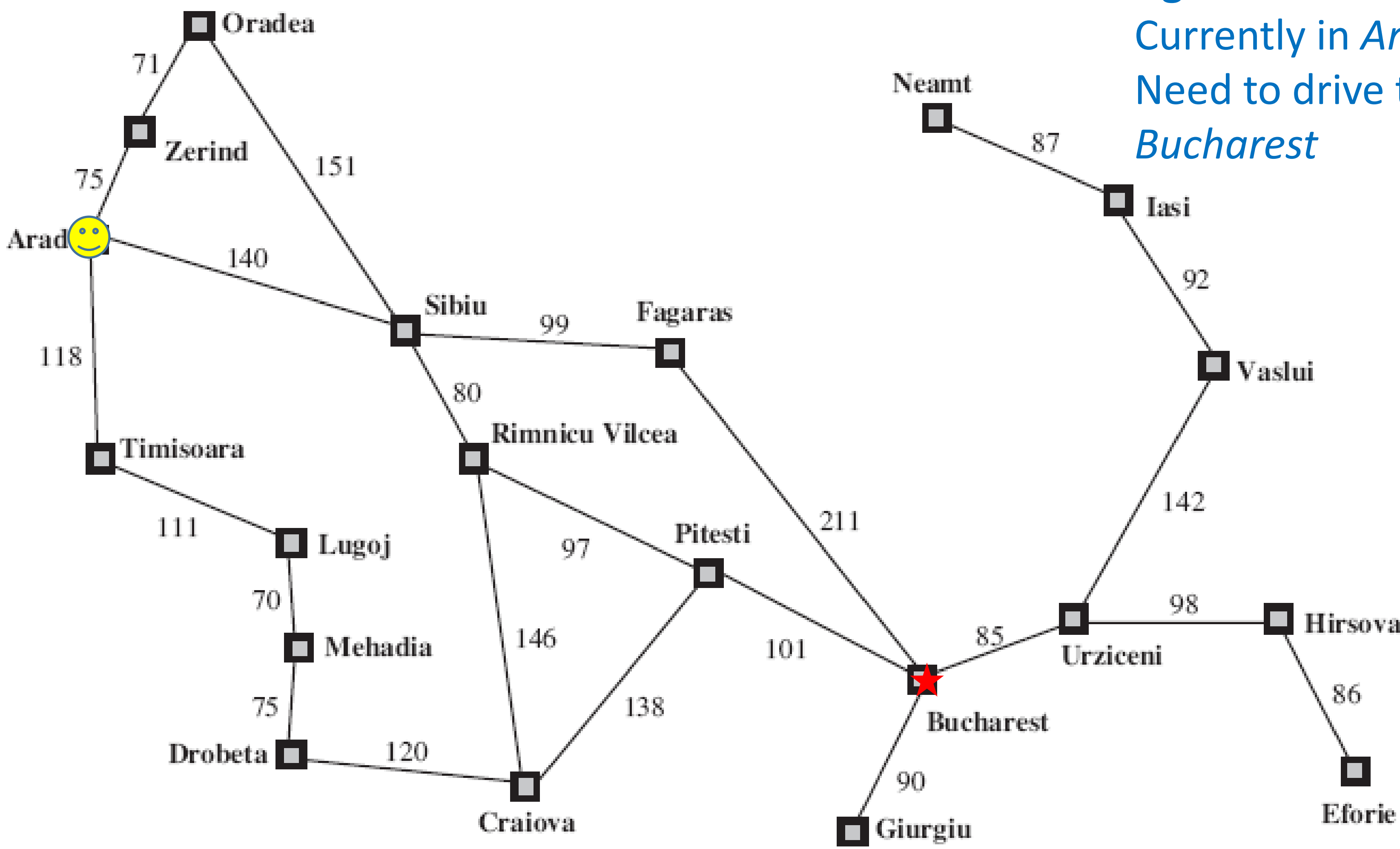
Solving Problems by Searching

CSEN266

Artificial Intelligence

Navigation in Romania



Agent:

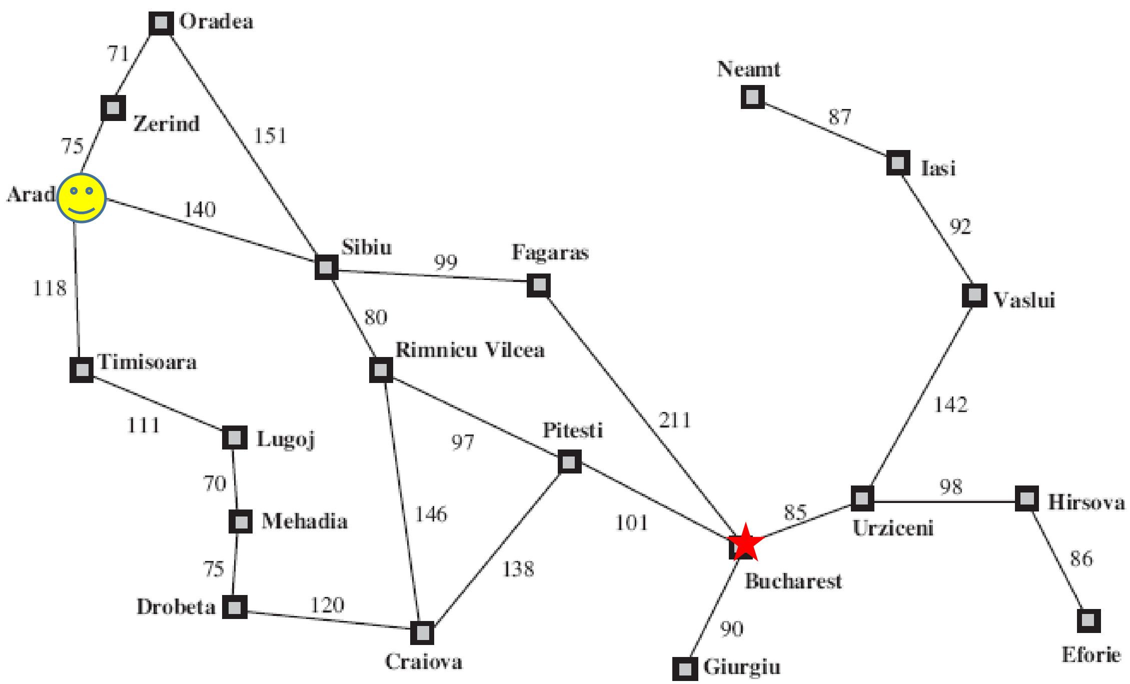
Currently in *Arad*;

Need to drive to

Bucharest

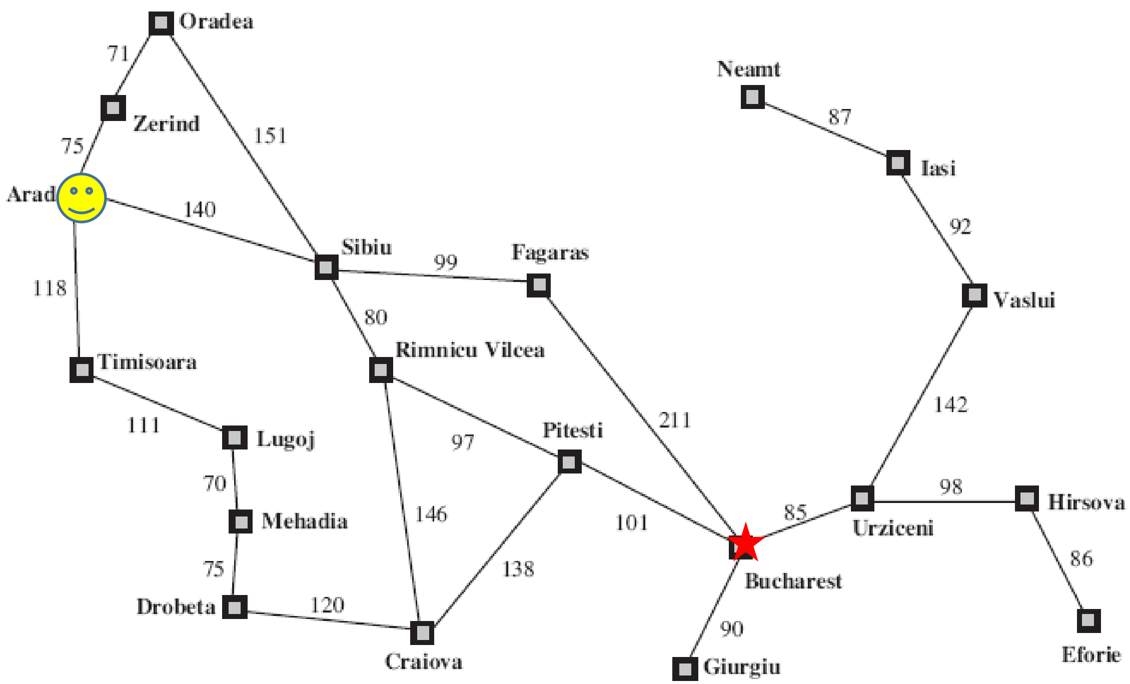
The **goal** of the problem?

- Get to Bucharest from Arad



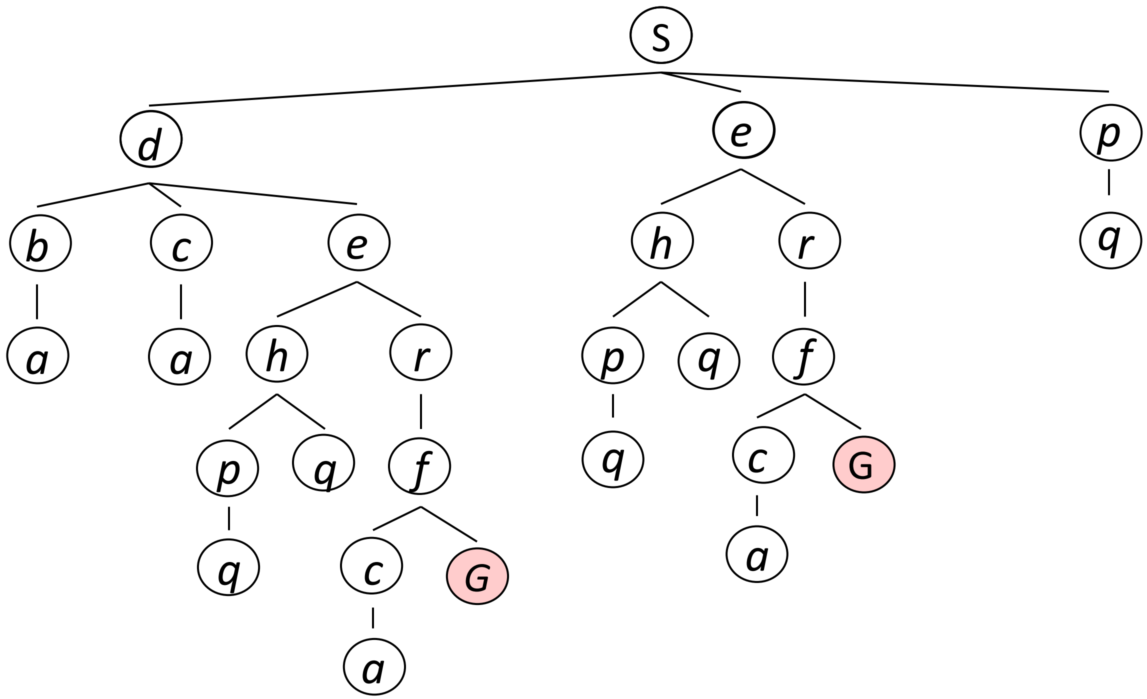
How to achieve the goal?

- Where to go from Arad? Sibiu, Timisoara, or Zerind?
- What actions to take? ⇒ Making decisions



Search Problem

- **Search:** look for a sequence of actions that reaches the goal



An Agent Wants to Do a Task

- **Search Phase**: the process of looking for the action sequence that achieves the goal

  - **Search Algorithm:** takes a problem as input and returns a solution

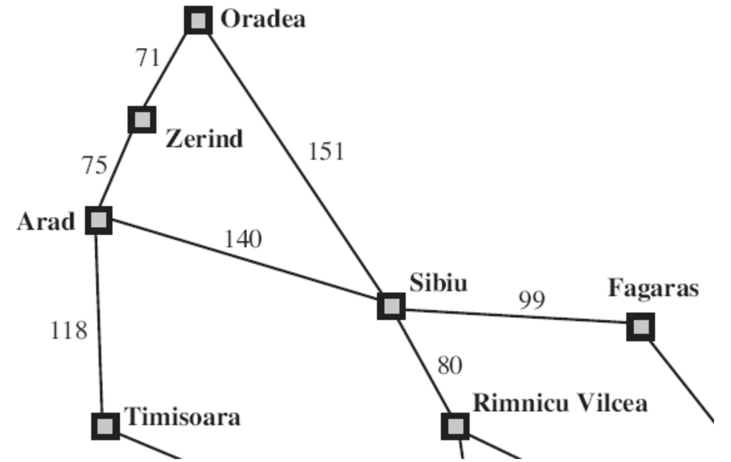
    ⬇

  - **Solution:** an action sequence that leads from the initial state to a goal state

- **Execution Phase:** the agent will carry out the actions recommended by the solution

Define a Search Problem

- Five components
  1. **States**
     e.g. *In(Arad), In(Sibiu), In(Fagaras)*
     Initial State: *In*(*Arad*)
     Goal State: *In*(*Bucharest*)



  2. **Actions** (available to the agent)
     - Given a state *s*, **Actions(*s*)** returns the set of actions that can be executed in *s*
     - **Actions(*In*(*Arad*)): {*Go*(*Sibiu*), *Go*(*Timisoara*), *Go*(*Zerind*)}**

Define a Search Problem

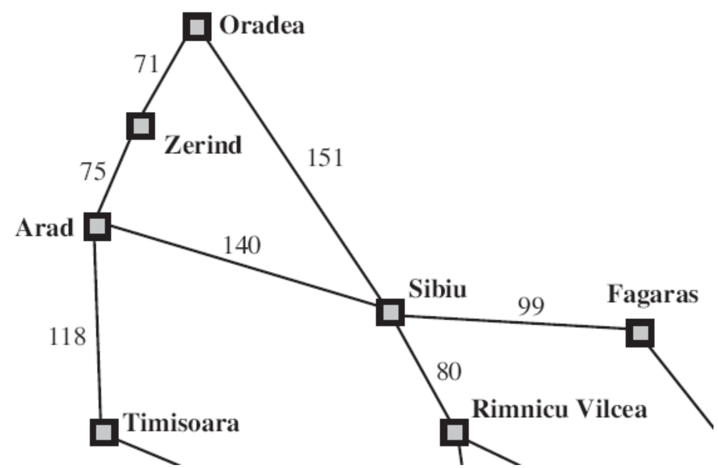
- Five components

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*

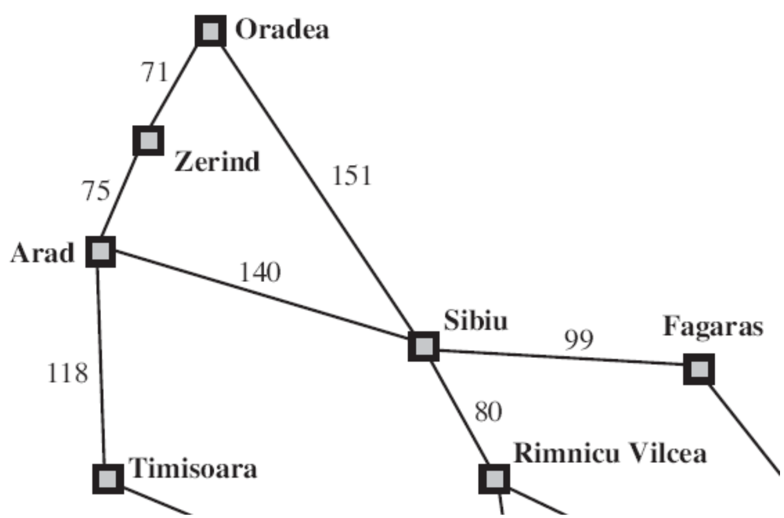
- **Result(In(Arad), Go(zerind)) = In(Zerind)**



Define a Search Problem

- Five components

4. **Goal test:** a function that verifies 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'***

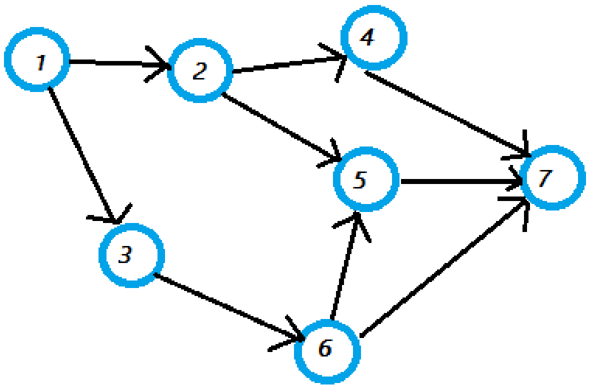


Define a Search Problem

- Additional Concepts

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

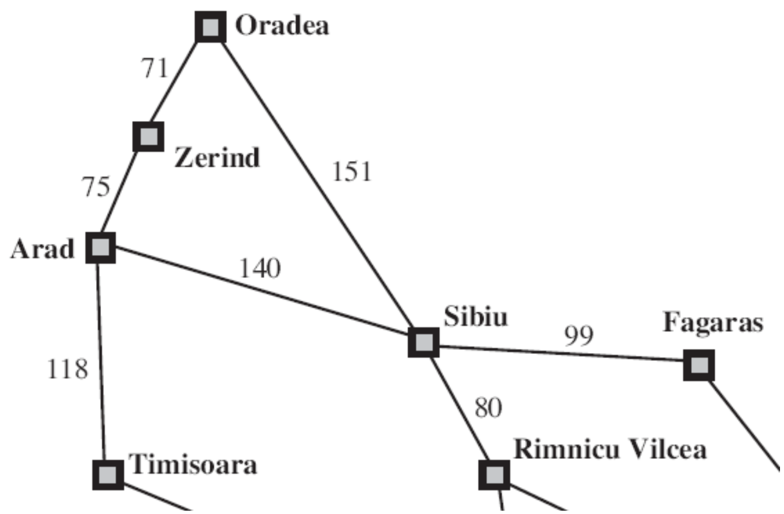
  - A directed graph (e.g. the map of Romania)
    - Nodes: the states
    - Links: actions



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

The Solution of a Search Problem

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



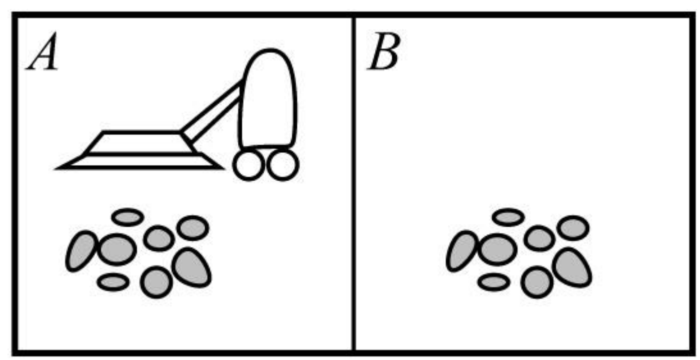
Vacuum World

- States
  - 2 agent locations (squares), each location might or might not contain dirt

    e.g. [‘Dirty’, ‘Dirty’, ‘A’] represents: square A is dirty, square B is dirty, the robot is currently in square A.

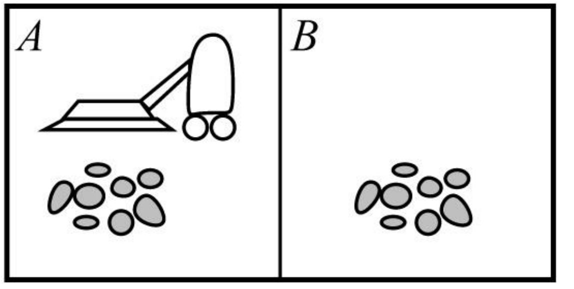
  - $2 \times 2^2 = 8$ states
  - If n locations, then how many states?

    $n \times 2^n$ states



Vacuum World

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



- Transition Model
  - e.g. s= [‘Dirty’, ‘Dirty’, ‘A’], a=‘Suck’

    s’ = Result(s, a) = [‘Clean’, ‘Dirty’, ‘A’]

Vacuum World

The complete state space (and transition model)

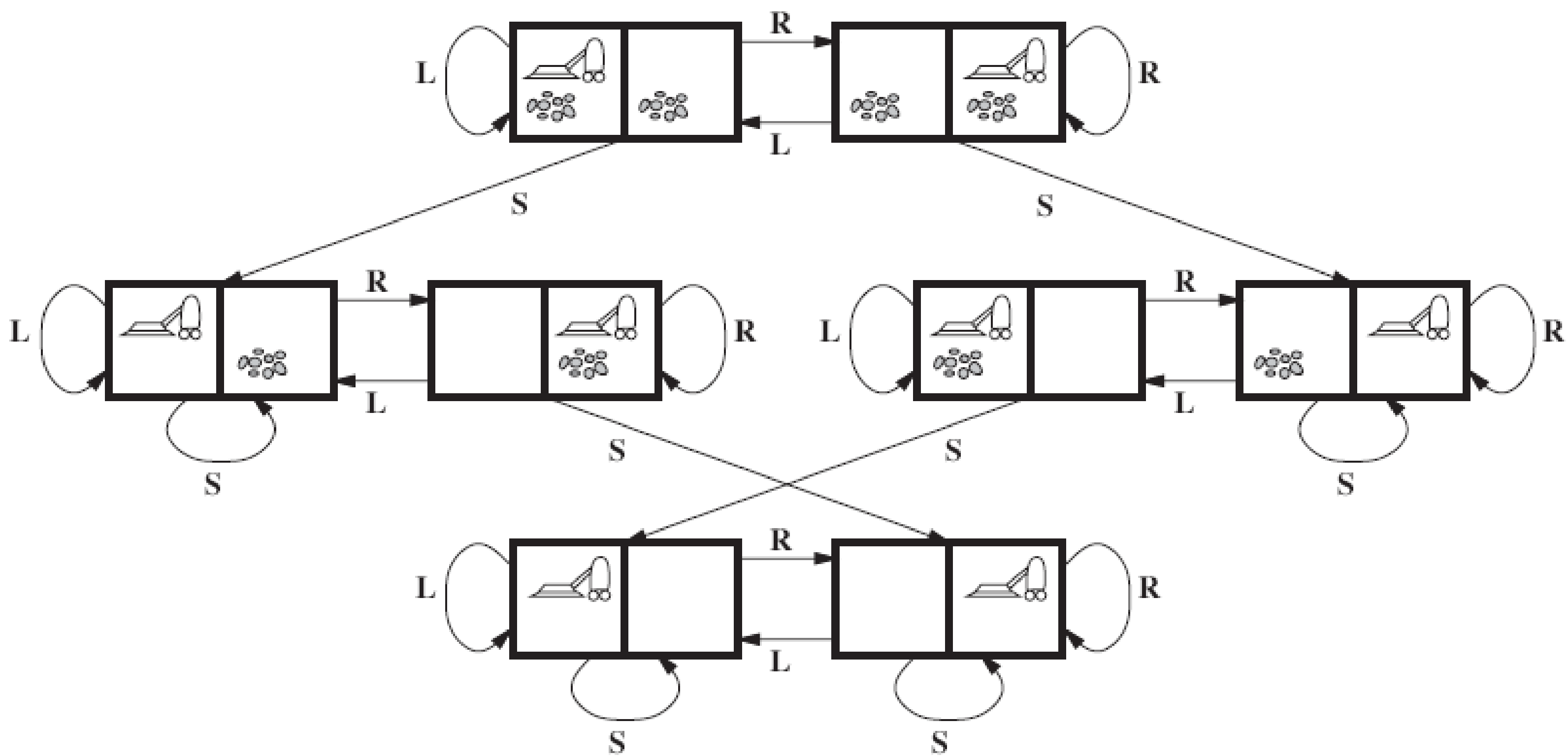


Figure 3.3 FILES: figures/vacuum2-state-space.eps (Tue Nov 3 16:24:01 2009). The state space for the vacuum world. Links denote actions: L = *Left*, R = *Right*, S = *Suck*.

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

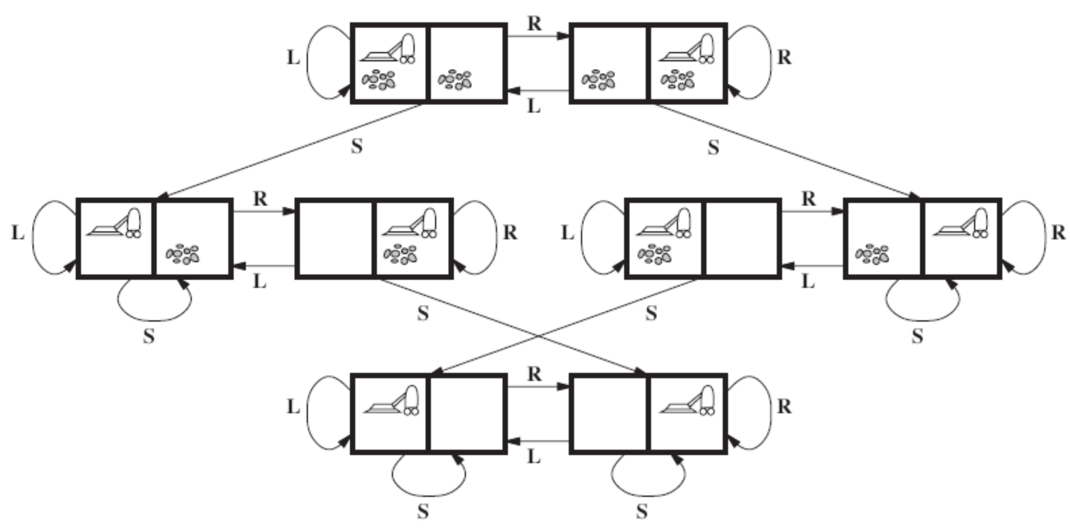
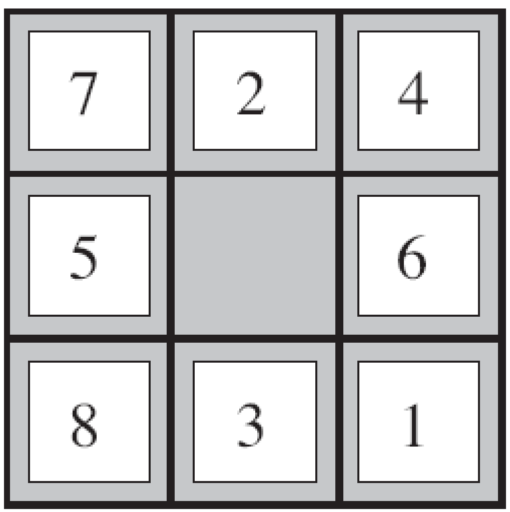


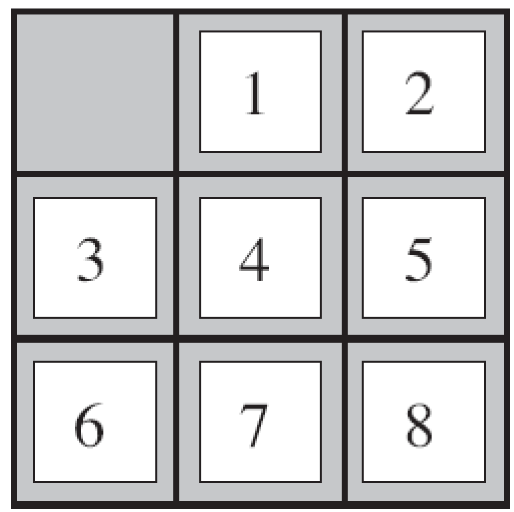
Figure 3.3 FILES: figures/vacuum2-state-space.eps (Tue Nov 3 16:24:01 2009). The state space for the vacuum world. Links denote actions: L = *Left*, R = *Right*, S = *Suck*.

The 8-puzzle

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



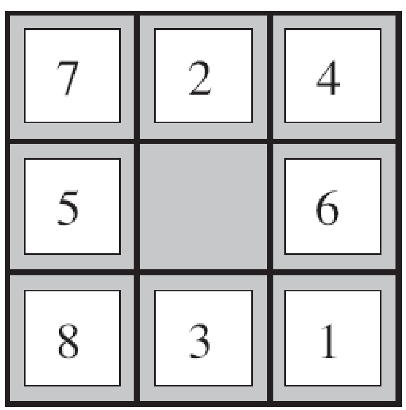
Start State



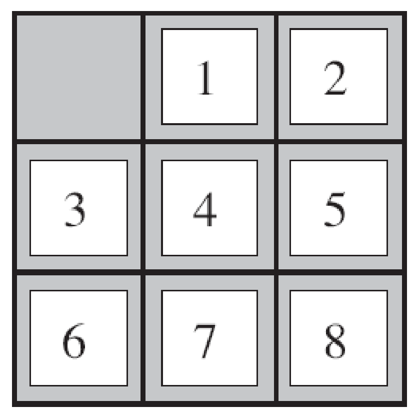
Goal State

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



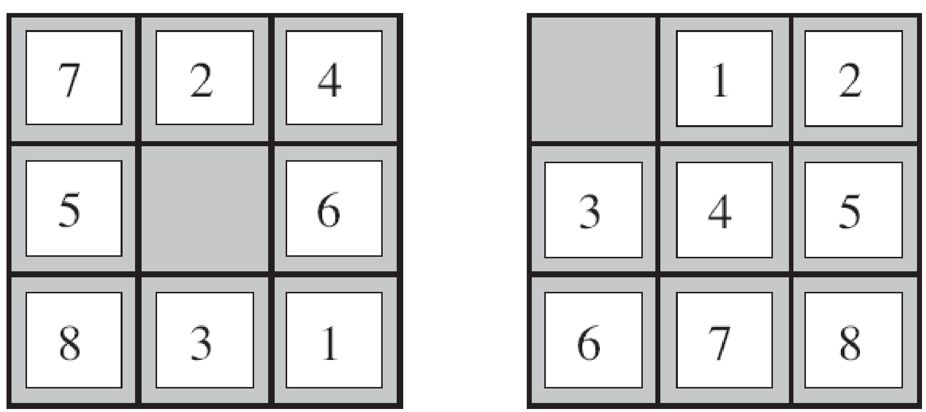
Start State



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.

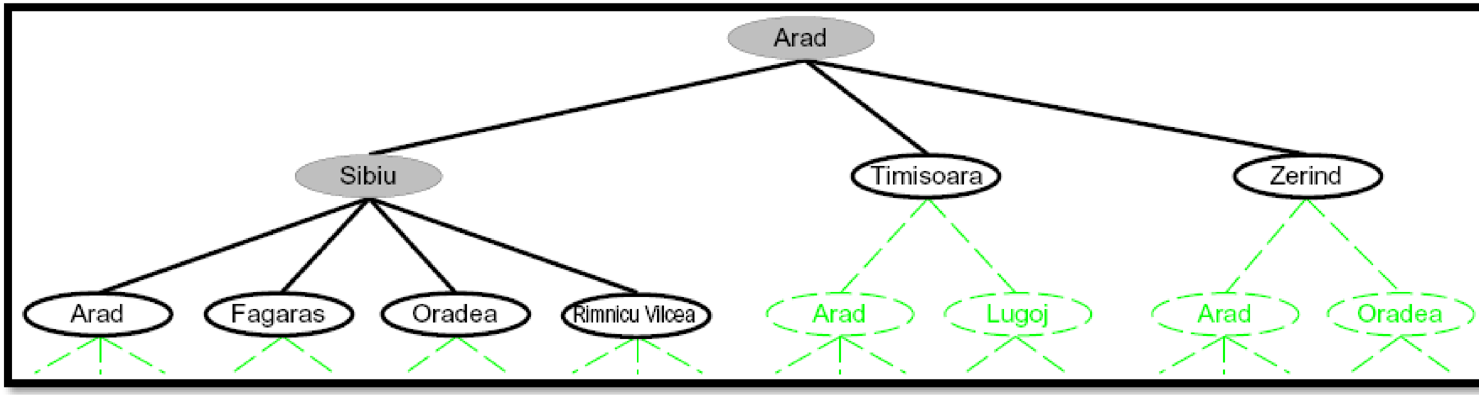
$$\left\lfloor \sqrt{\sqrt{\sqrt{\sqrt{\sqrt{(4!)!}}}}} \right\rfloor = 5$$

- Transition model?
  - Definition of these math operations
- Goal test?
  - State is the desired positive integer

- Infinite state space

Search Algorithms

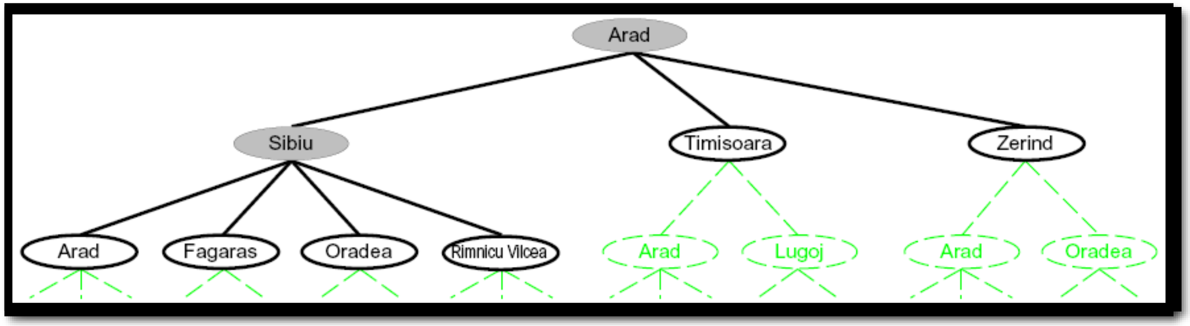
- Look for a sequence of actions that achieve the goal
- **Search Tree**

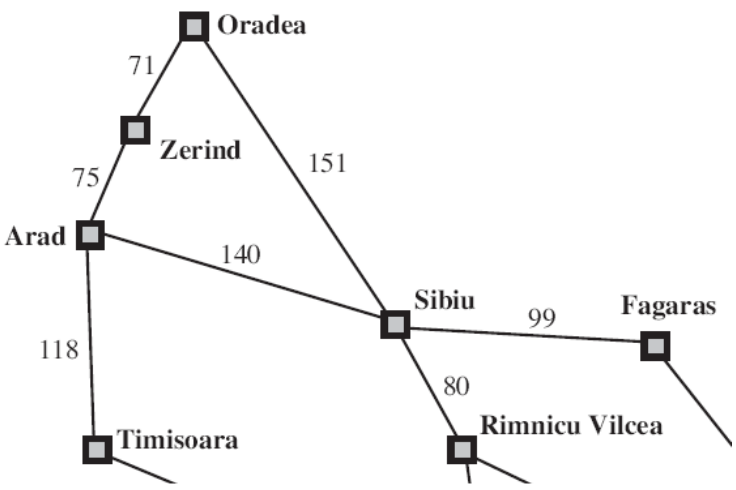


  - **Nodes:** states in the state space
    - Parent node, child nodes
    - Leaf node: a node with no children
  - **Root Node:** initial state
    - In(Arad)
  - **Branches:** actions

Search Algorithms

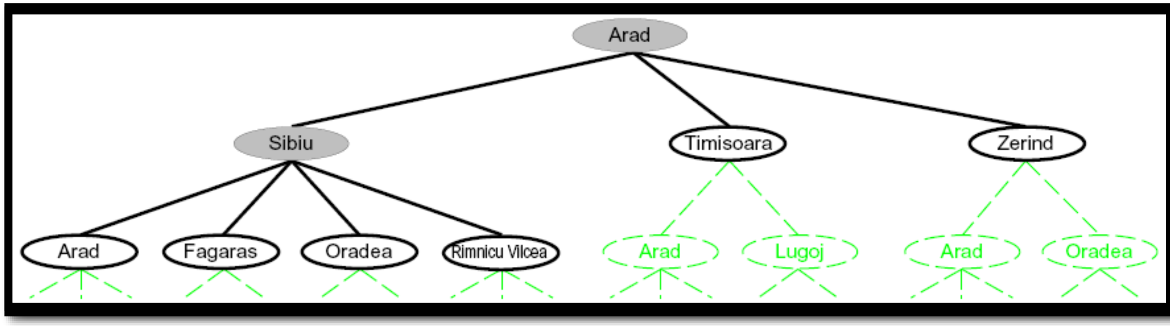
- Expand a node and generate a new set of nodes
- How to choose which node to expand?

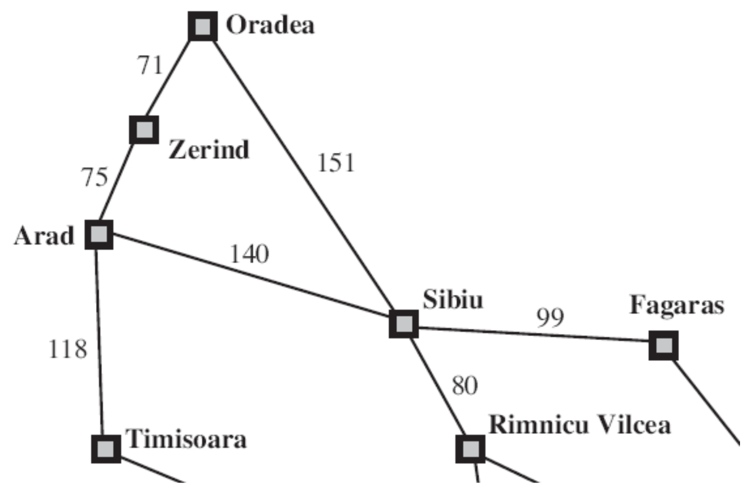




Search Algorithms

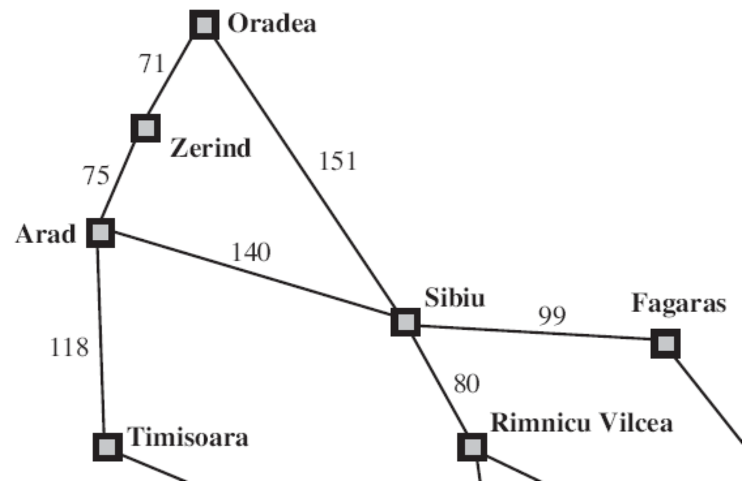
- Frontier nodes: the candidate nodes to be expanded

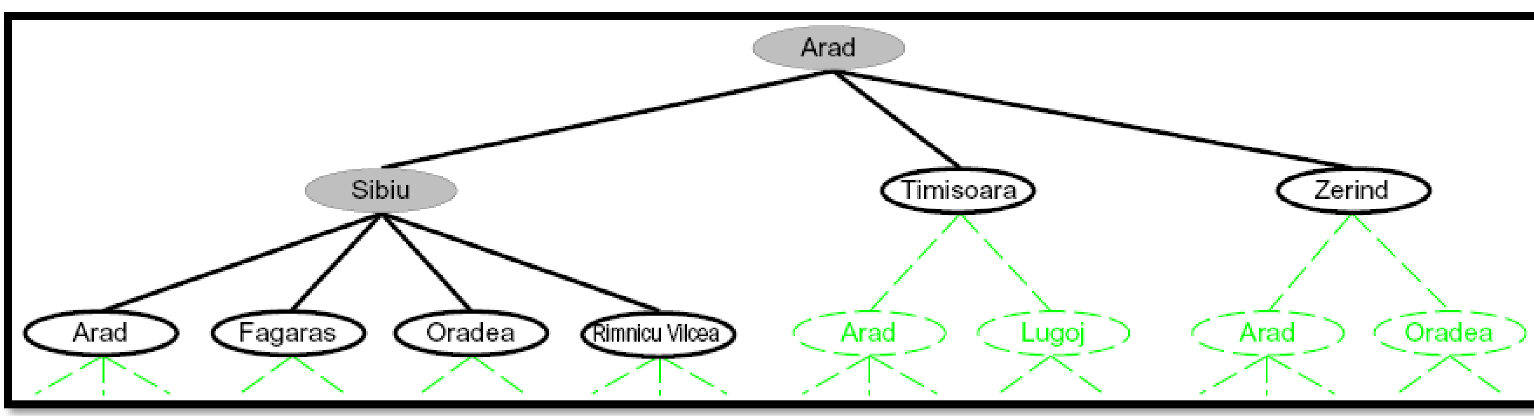




Tree-Search Scheme

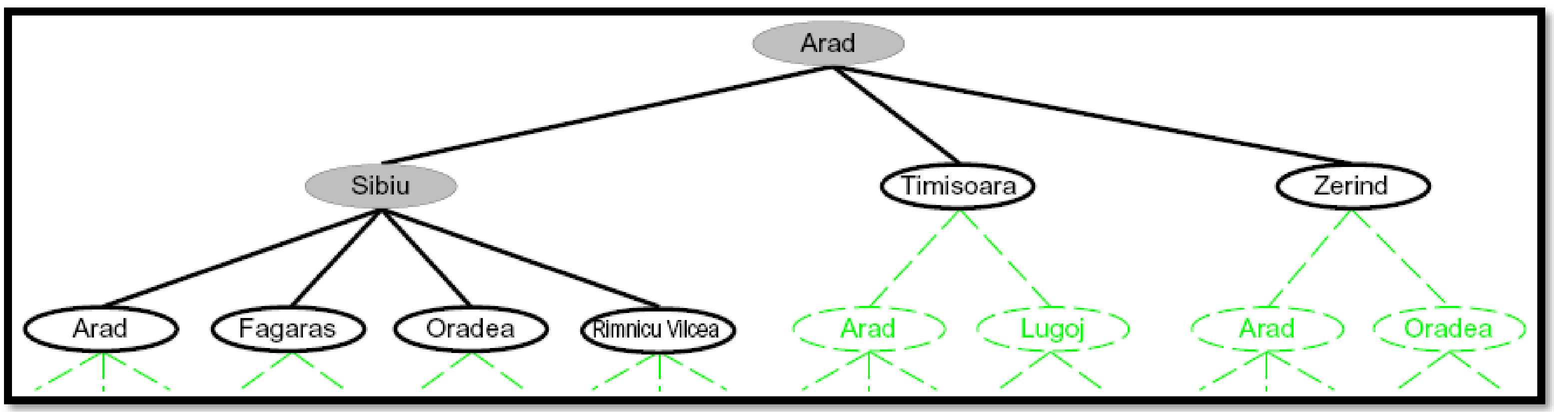
- The same state can be visited repeatedly
  - Arad-Sibiu-Arad
  - Loopy path
    - The complete search tree for Romania is *infinite*
    - Can cause certain algorithms to fail





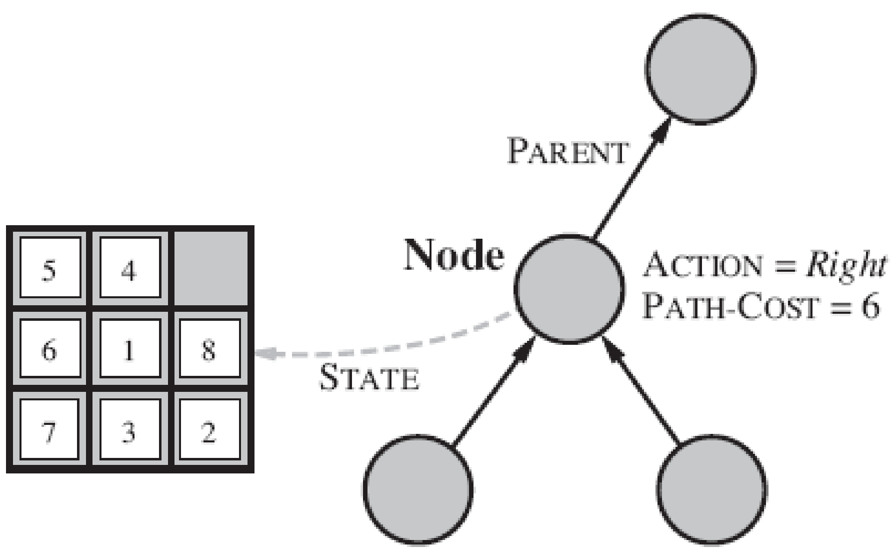
Graph-Search Scheme

- Never expands the same state twice



Data structure for node n on the search tree

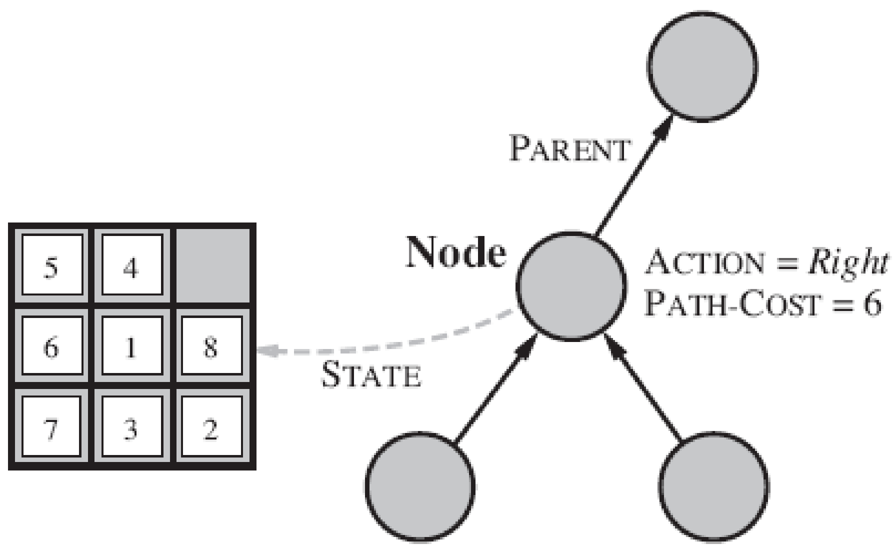
- Data structure: keeps track of the search tree
- n.State: the state in the state space to which the node corresponds.
- n.Parent: the node in the search tree that generated this node.
  - Arrows point from child to parent



Data structure for node n on the search tree

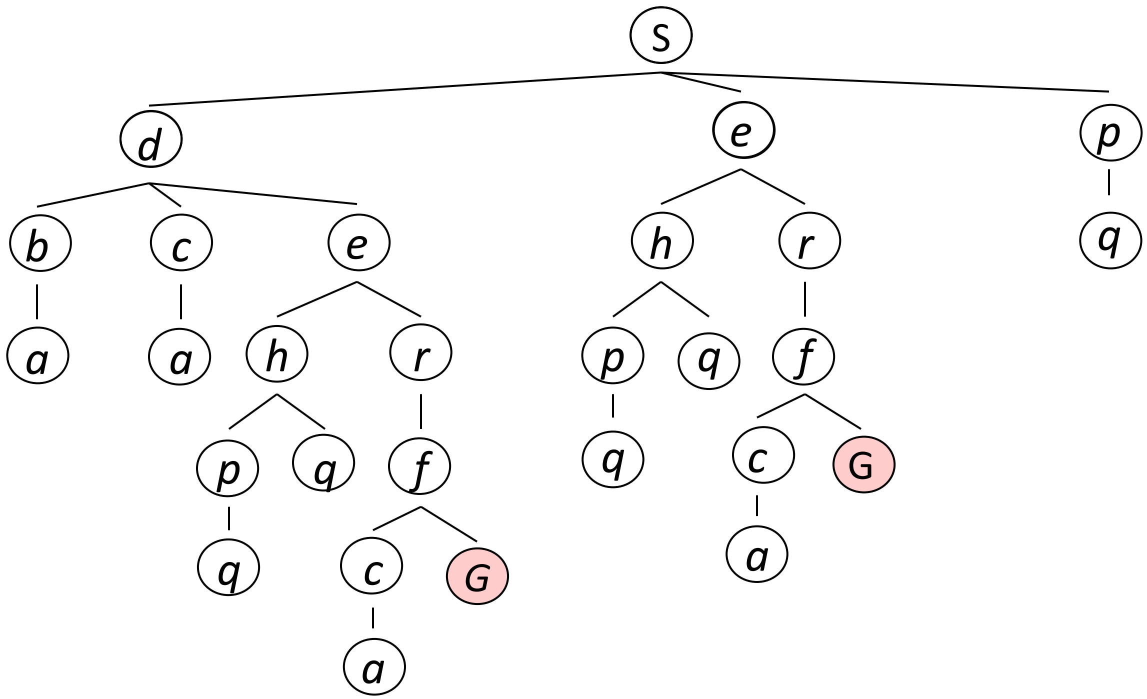
- n.Action: the action that was applied to the parent to generate the node.
  - e.g. n.Action: movement of the blank tile

    Draw the state of the parent node?
- n.Path-Cost $g(n)$: the cost of the path from the initial node to node n, as indicated by the parent pointers.



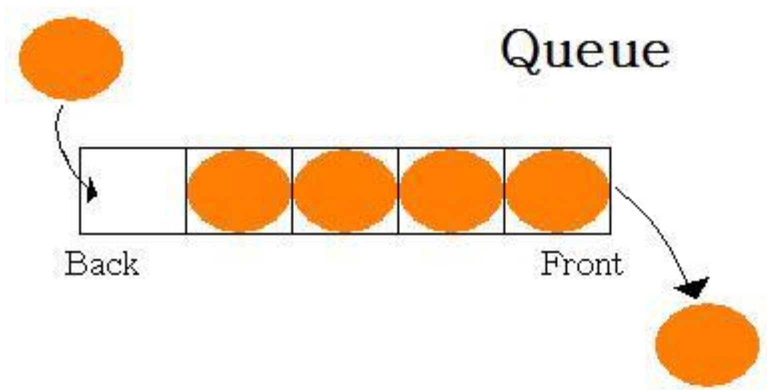
Data structure for node n on the search tree

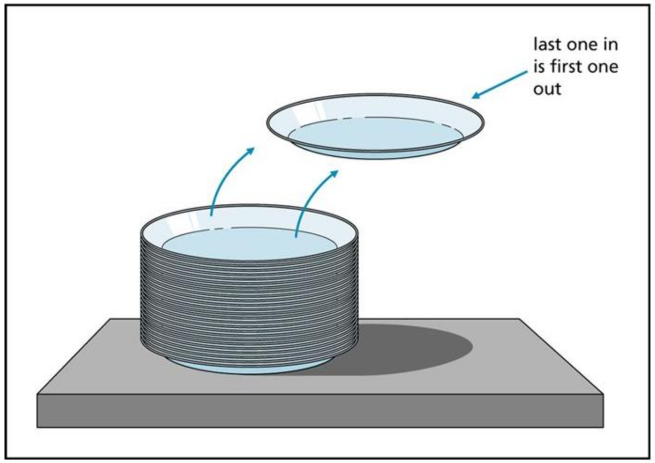
- **Illustrate:** n.State, n.Parent, n.Action, n.Path-Cost=$g(n)$



Frontier Set

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





Performance Measurement

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

- Optimality: Does the strategy find the optimal solution (w.r.t some performance measure)?
  - e.g. The path cost of the solution found (such as the total length of the path in kilometers)

Performance Measurement

- Time complexity: How fast does the algorithm find a solution?
  - Measured in terms of the number of nodes expanded/visited/explored during the search process

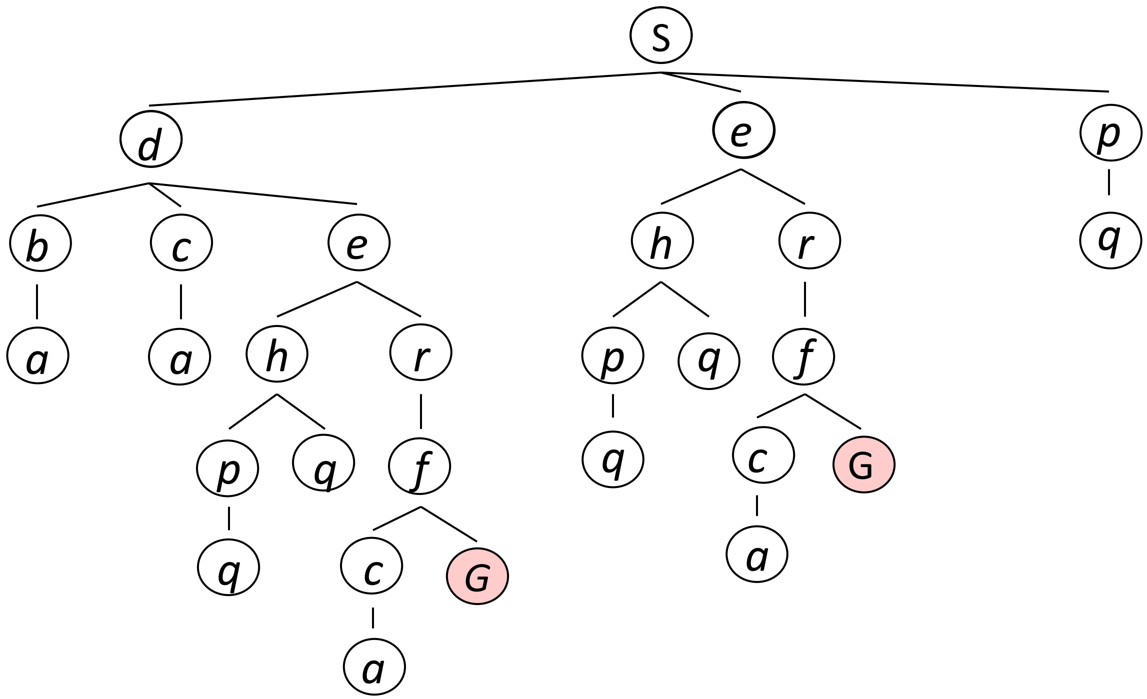
- Space complexity: How much memory is needed to perform the search?
  - Measured in terms of the maximum number of nodes stored in memory during the search process

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

Parameters

- What are b, d, and m for this search tree?

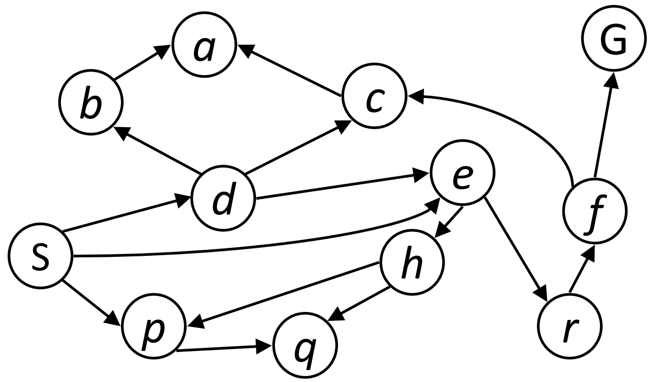


Uninformed Search

- Also called: Blind Search
- The search strategy
  - does not know which non-goal states are better than other non-goal states
  - can only
    - Generate successors
    - Distinguish a goal state from a non-goal state

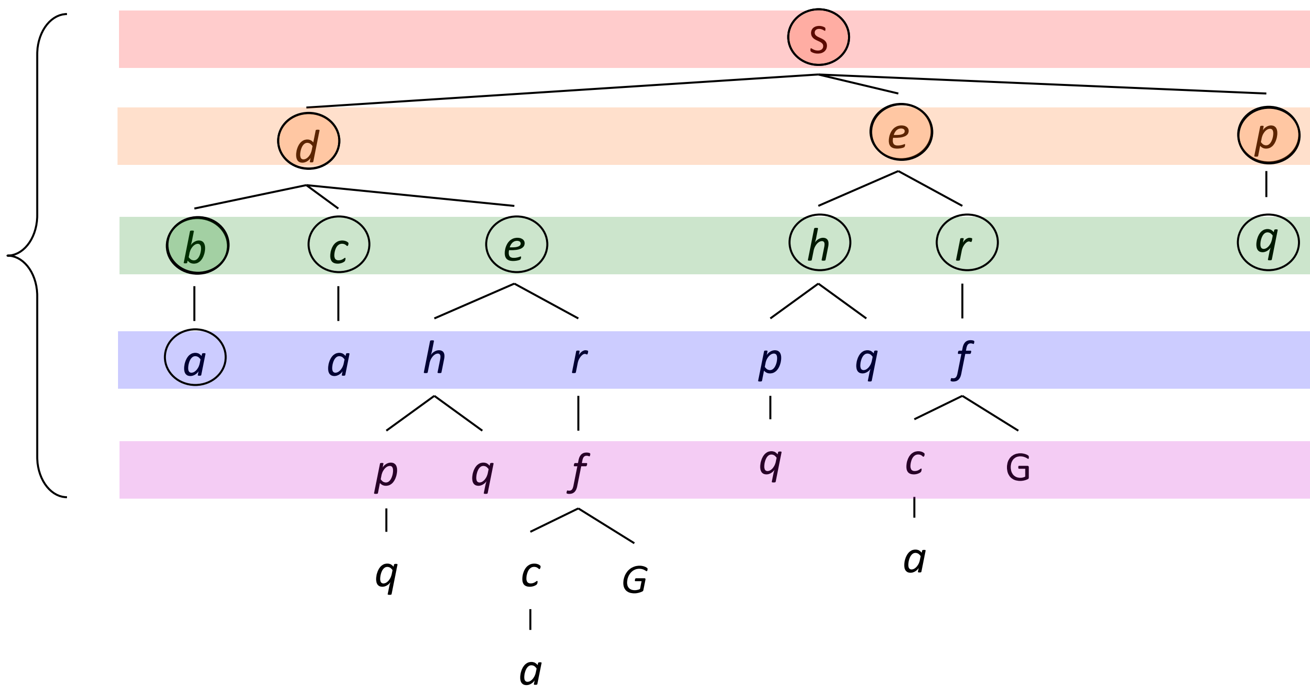
- Different uninformed search strategies
  - distinguished by the **order** in which nodes are expanded

Breadth-First Search



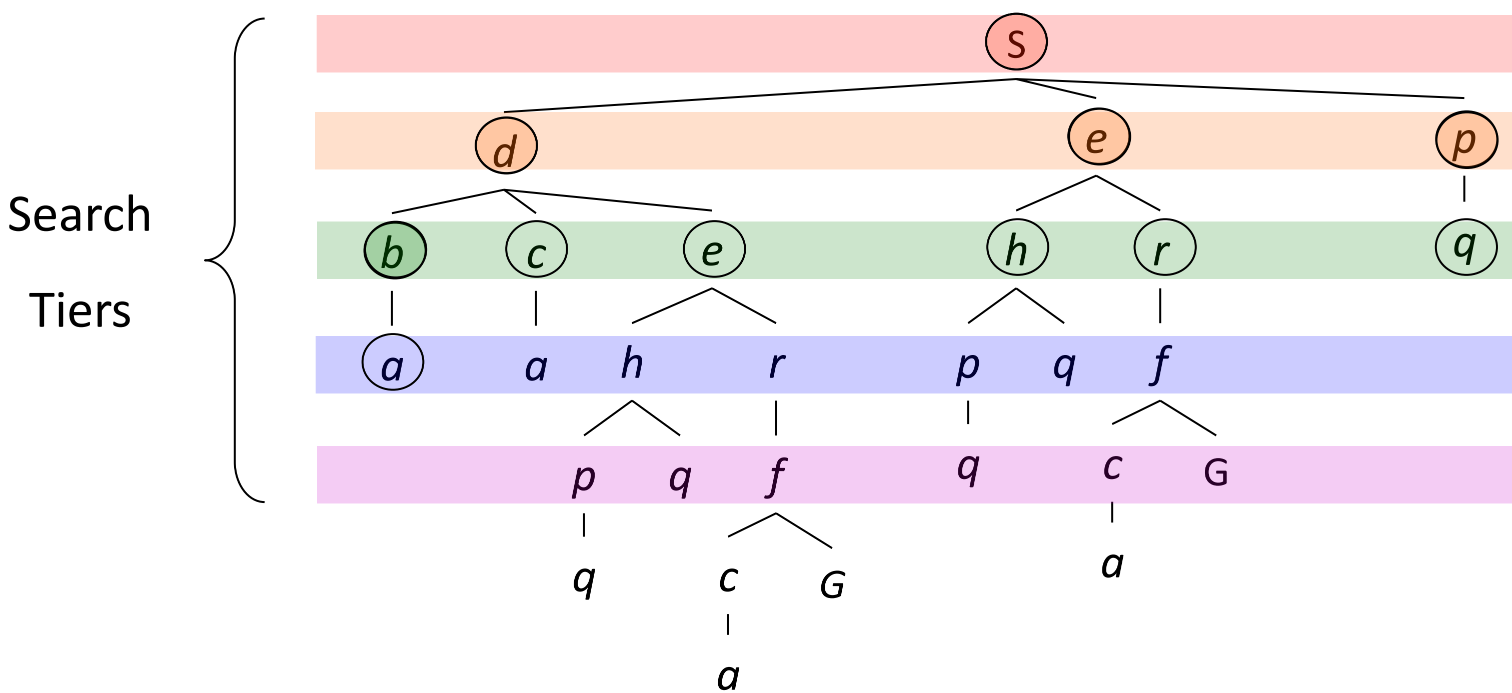
Initial State: S
Goal State: G
Explored: color circled
Frontier: white circled
Unexplored: uncircled

Search Tiers



Breadth-First Search

- Shallower nodes are expanded before deeper nodes
- Achieved by using a FIFO queue for the frontier nodes



Breadth-First Search

- Optimality?
  - In terms of solution path cost
  - Optimal in the sense that it always finds the shallowest goal node

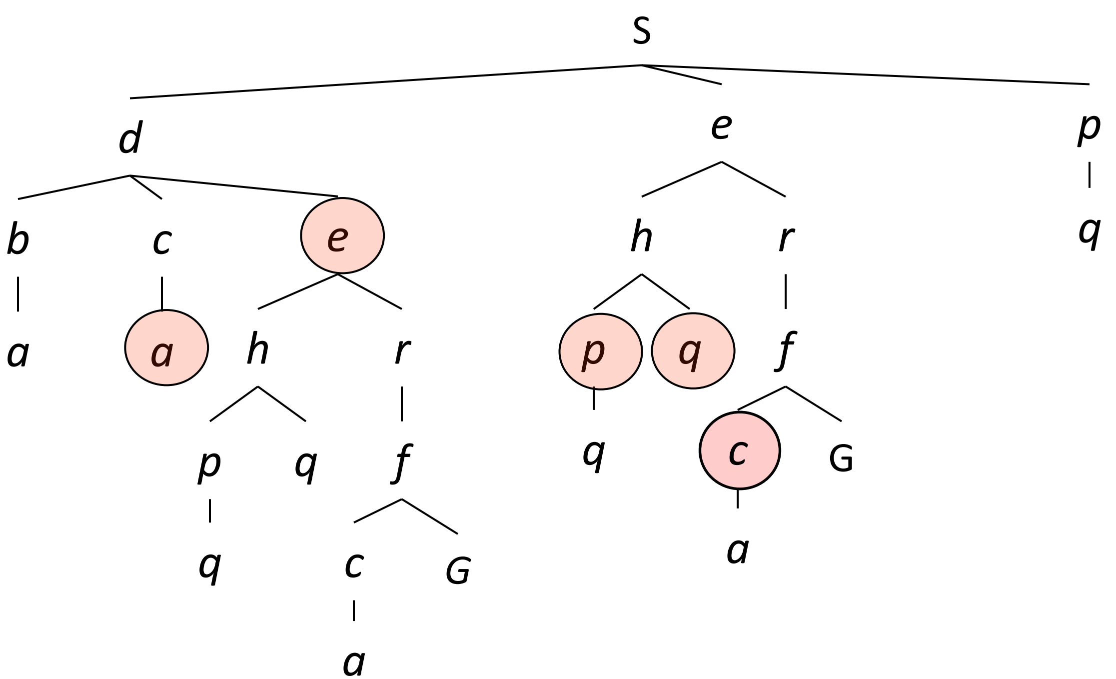
- Complete?
  - Yes if the branching factor b is finite

Graph-Search Scheme

- Idea: never visit the same state twice
- How to implement:
  - Construct a search tree
  - Keep a set of visited states
  - Expand the search tree node-by-node as in a tree search strategy, but…
  - Before expanding a node, check whether its state has been visited before
    - If yes, skip the node;
    - If no
      - add its state to the set of visited states
      - expand the node and generate the successors

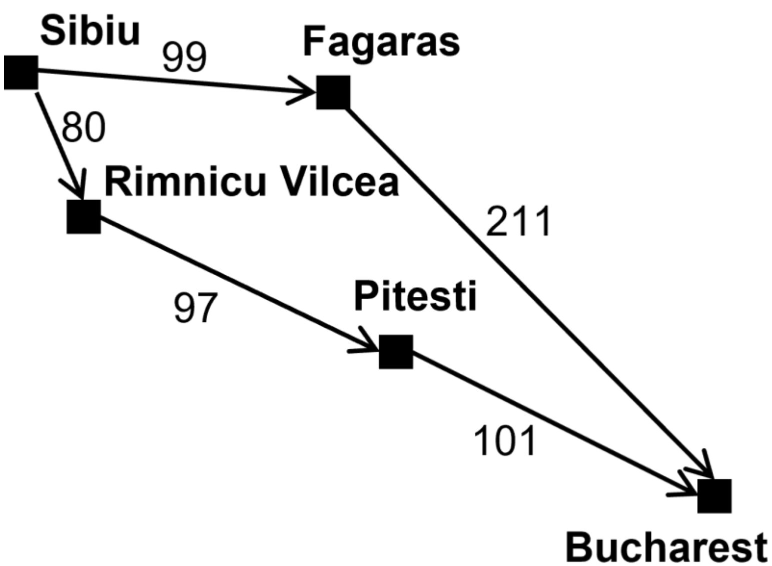
Graph-Search Scheme

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



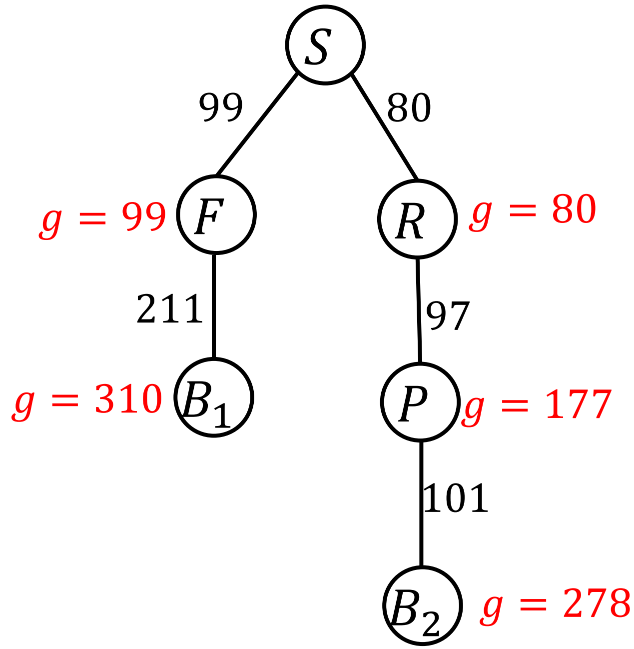
Uniform-cost Search

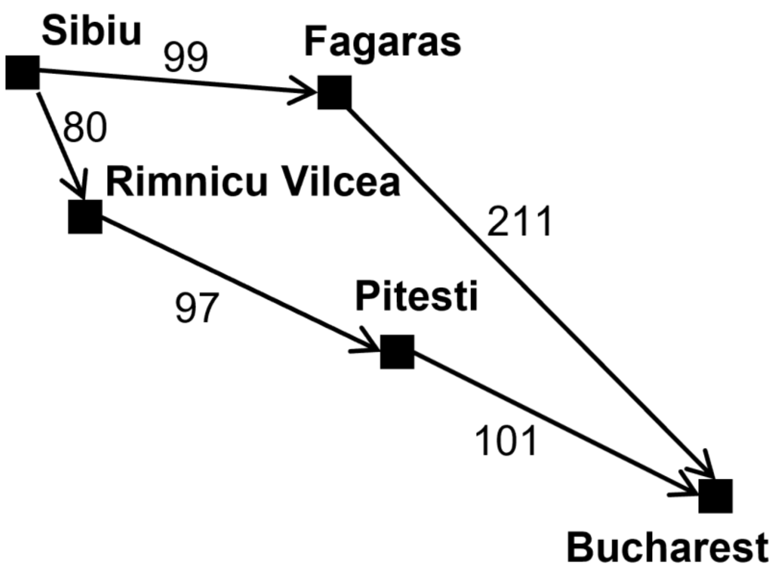
- Expands the node n with the *lowest path cost* $g(n)$
  - Done by storing the frontier as a priority queue ordered by g
  - *Path cost* $g(n)$: the cost of the path from the initial node to node n
- Example: get from Sibiu to Bucharest



Uniform-cost Search

Visited Nodes	Frontier
S	F, R
R	F, P
F	P, B_1
P	B_1, B_2
B_2	



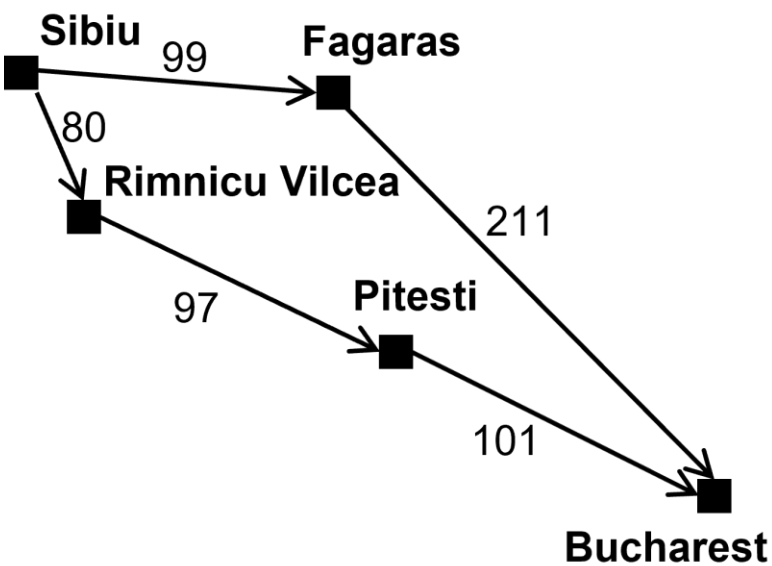


Solution path: $S \rightarrow R \rightarrow P \rightarrow B_2$

Solution path cost: 278

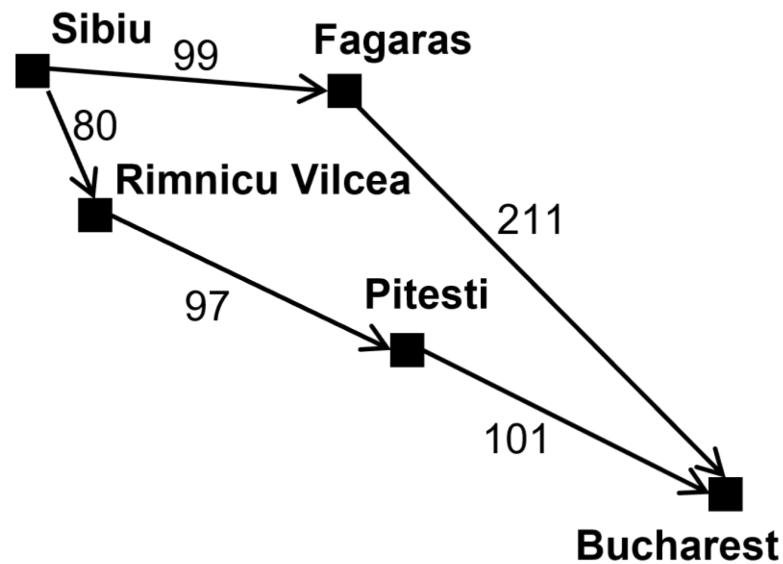
Uniform-cost Search

- Guided by path costs
- Does not care about the number of steps a path has



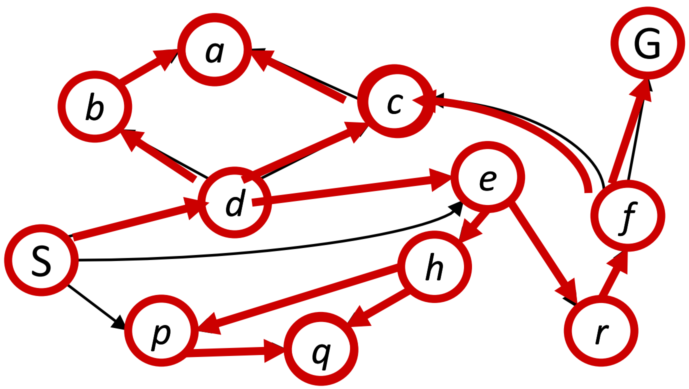
Uniform-cost Search

- 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



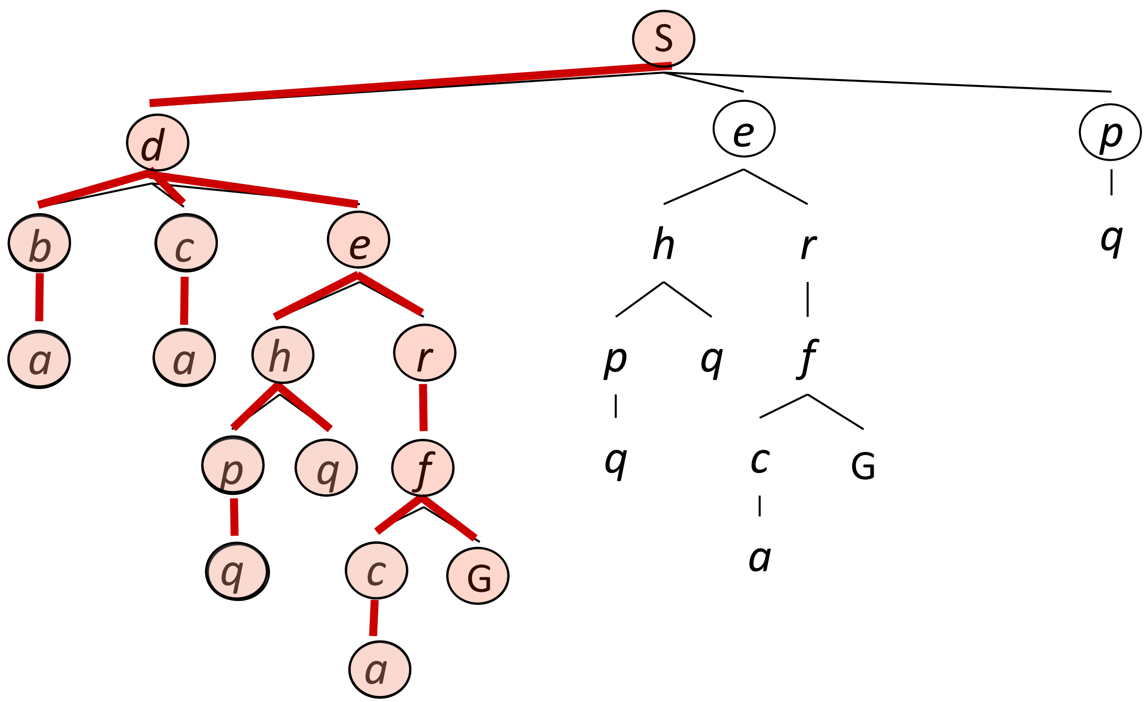
Initial State: S

Goal State: G

Explored: color circled

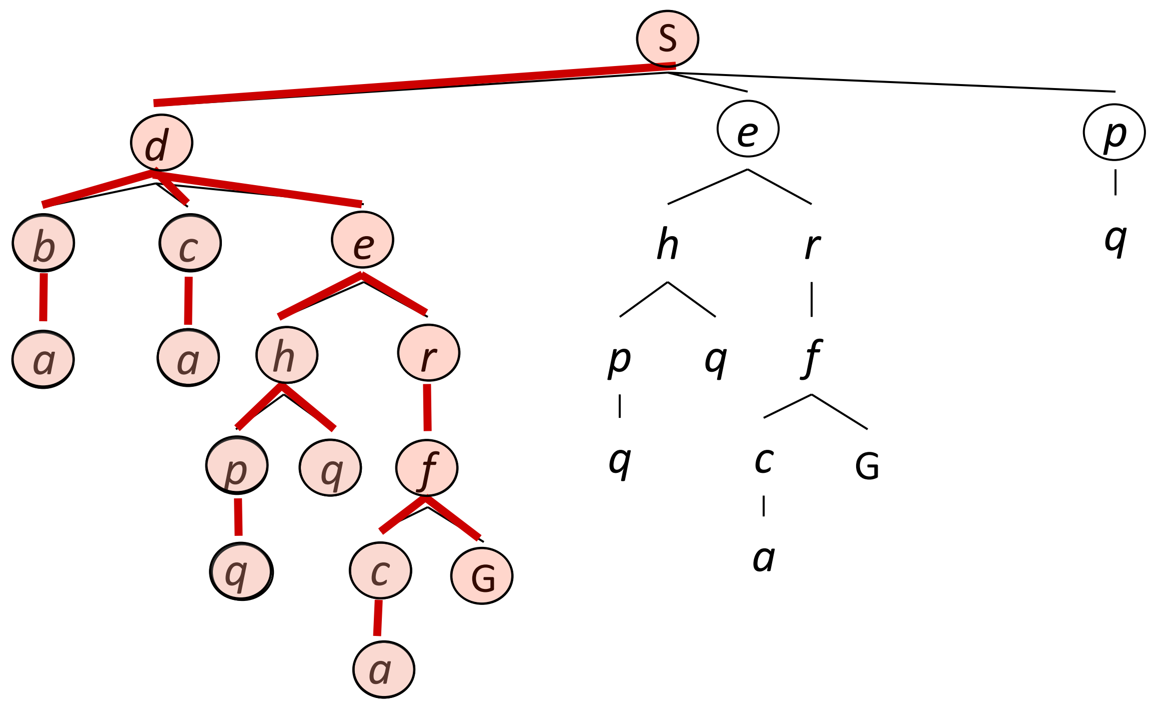
Frontier: white circled

Unexplored: uncircled



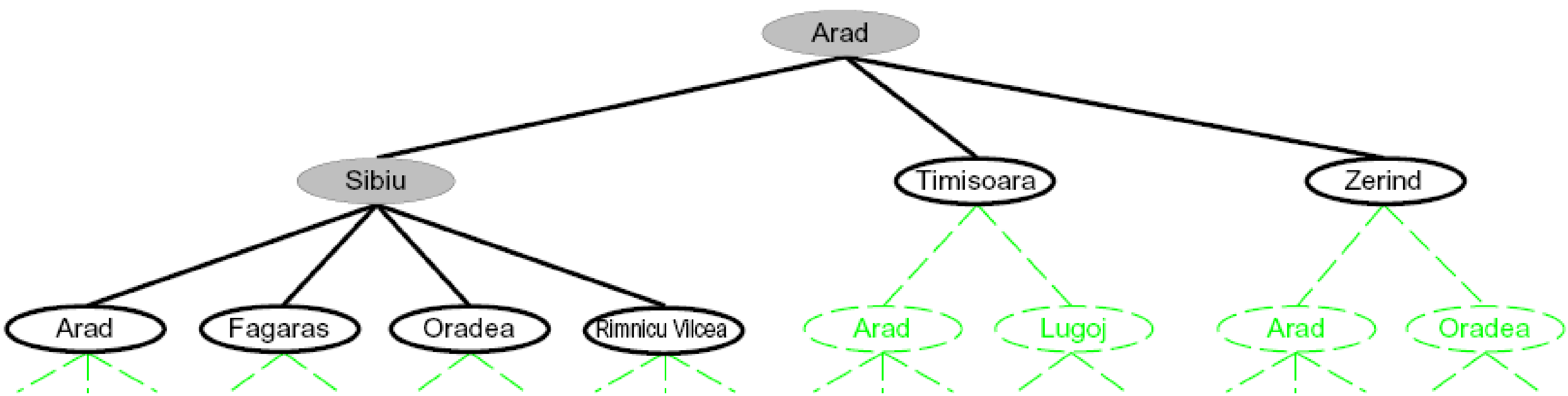
Depth-First Search

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



Depth-First Search

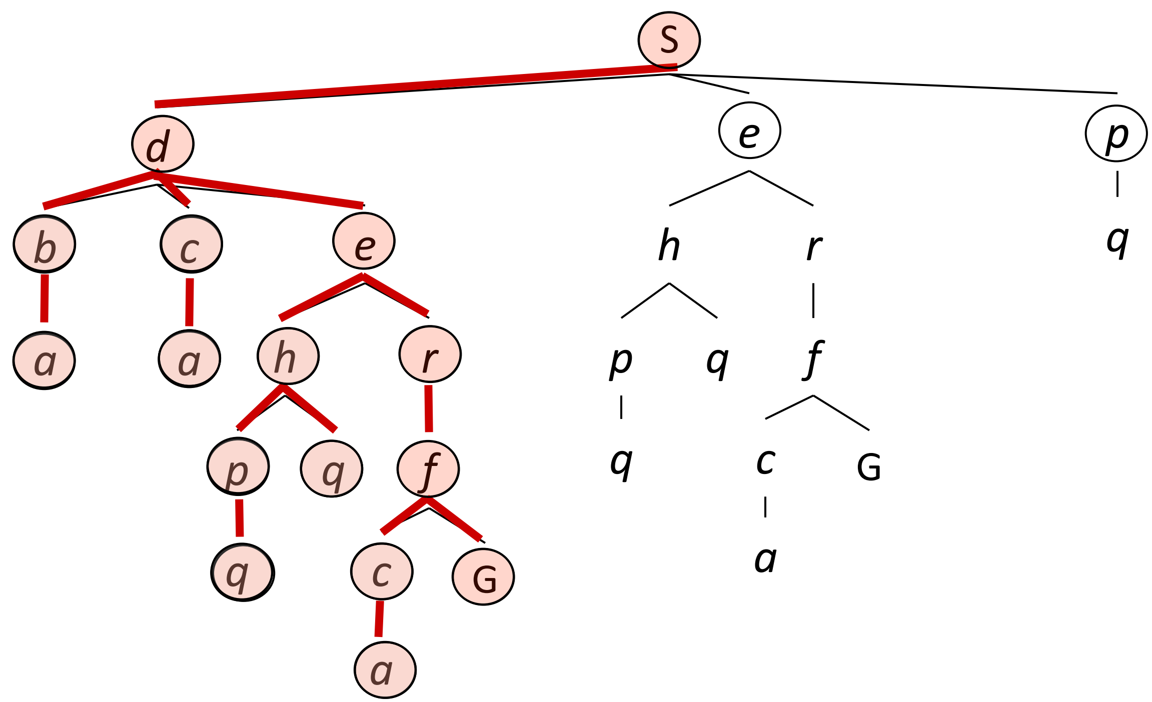
- Complete or not?
  - Depth-first Graph-search: Yes (avoids repeated states)
  - Depth-first Tree-search: No



- Arad-Sibiu-Arad-Sibiu loop forever!

Depth-First Search

- Optimal or not? (in terms of the cost of the solution path)



- Answer: No!
- The solution returned by the DFS will get to the goal state in 5 steps instead of 4 steps

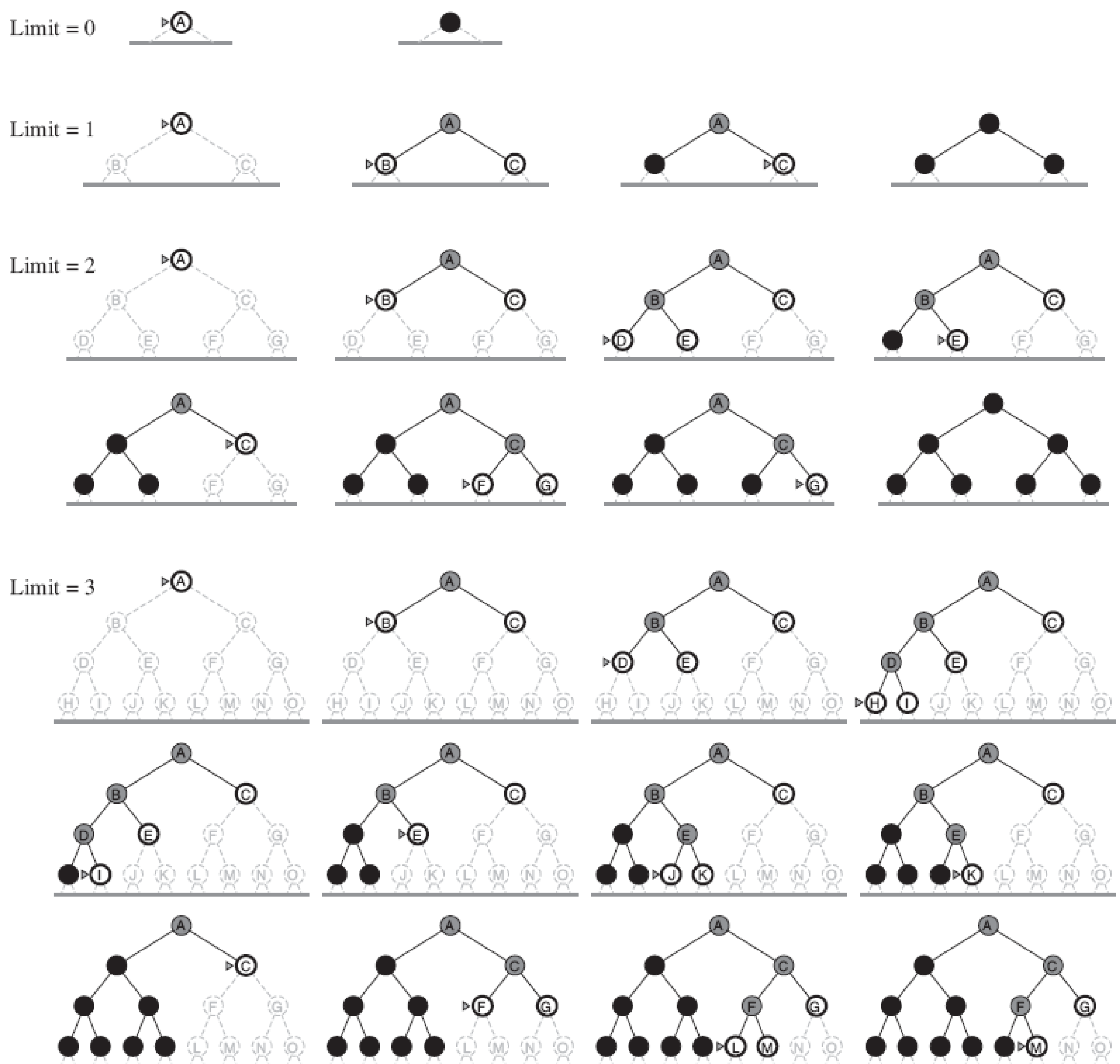
Depth-Limited Search

- 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.

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)

Iterative Deepening DFS

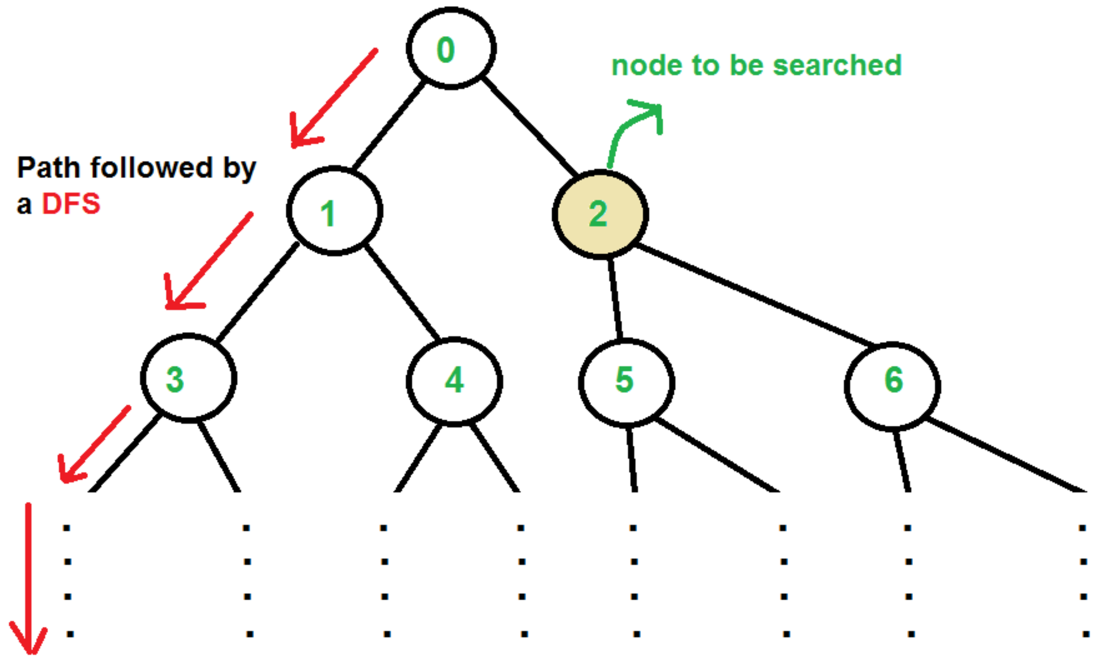


Four iterations of iterative deepening DFS on a binary tree

Suppose: M is the goal node

Black circles: nodes removed from memory

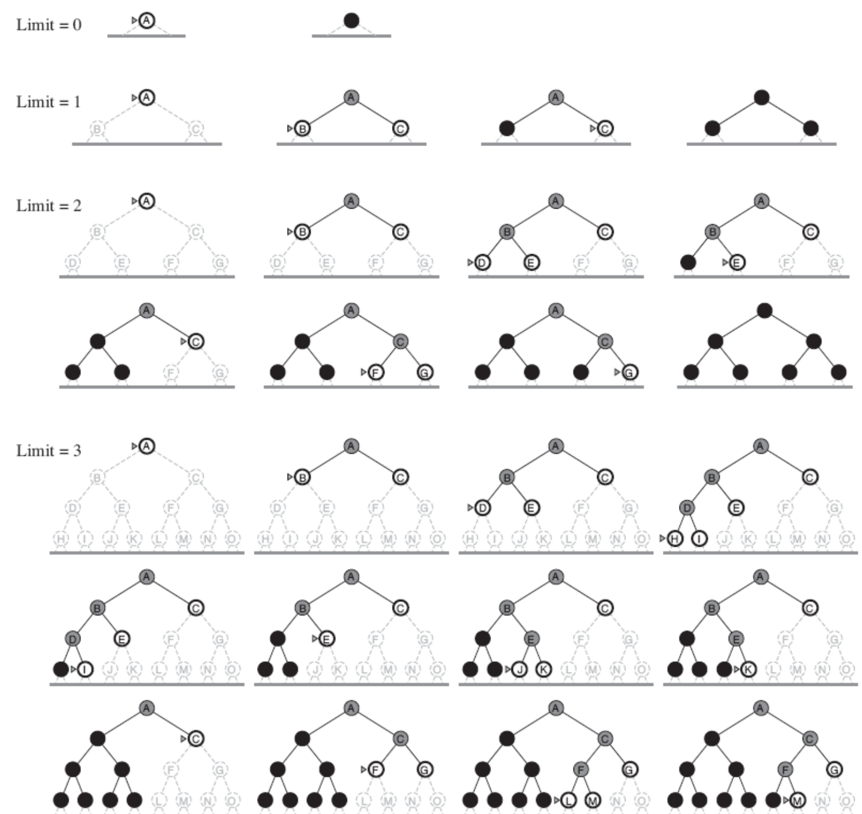
Iterative Deepening DFS



- 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.



Iterative Deepening DFS

- Combines the benefits of DFS and BFS
  - Benefit of BFS: optimal (shallowest solution)
  - Benefit of DFS: space complexity

Informed Search Strategies

- The search strategies have extra information regarding how “close” a node is to a goal node

- Can find solutions more efficiently than uninformed search strategies

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
  - 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}

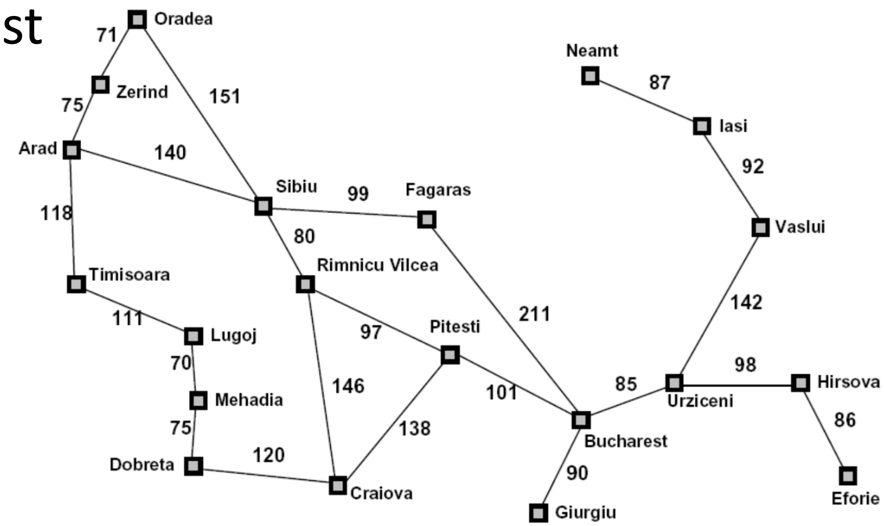
The straight-line distance to Bucharest

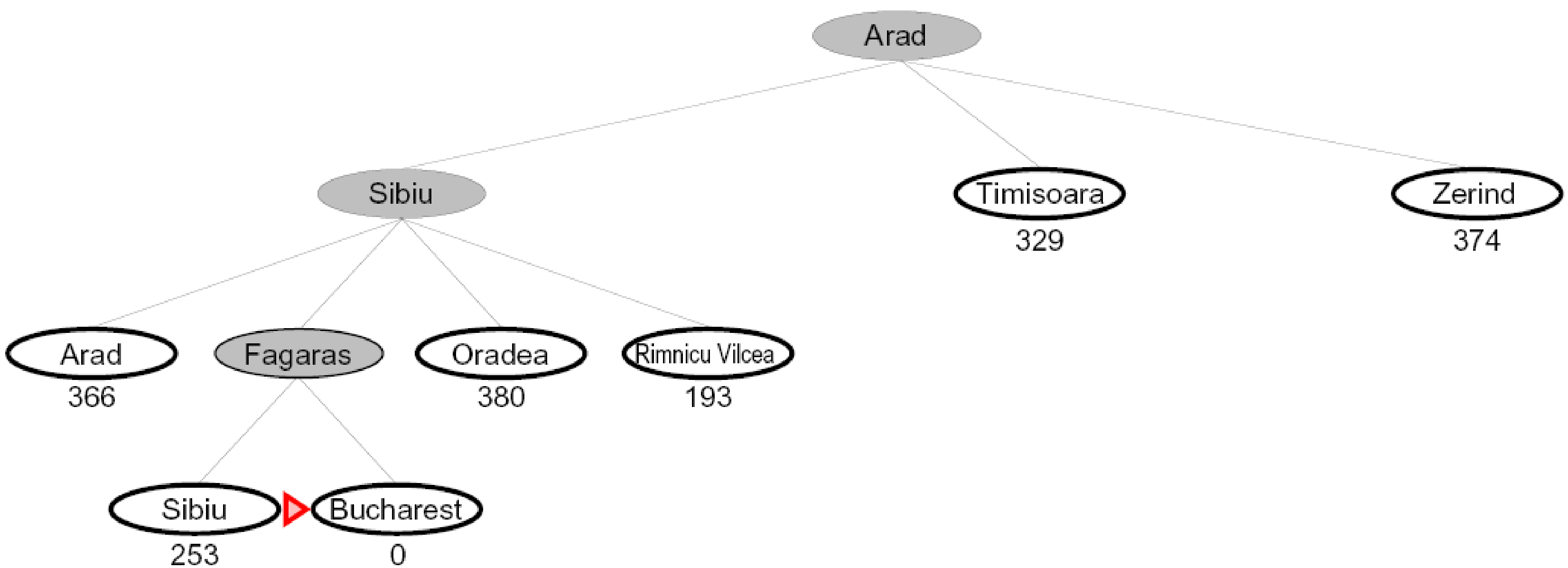
Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Drobeta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374

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

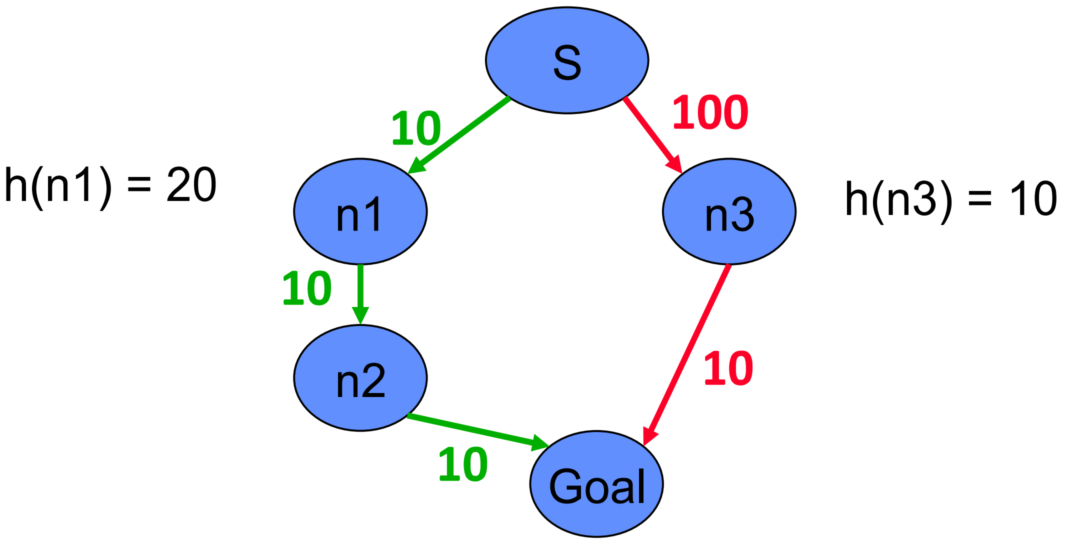
- Why is it called “greedy”?
  - At each step, it tries to get as close to the goal as it can

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

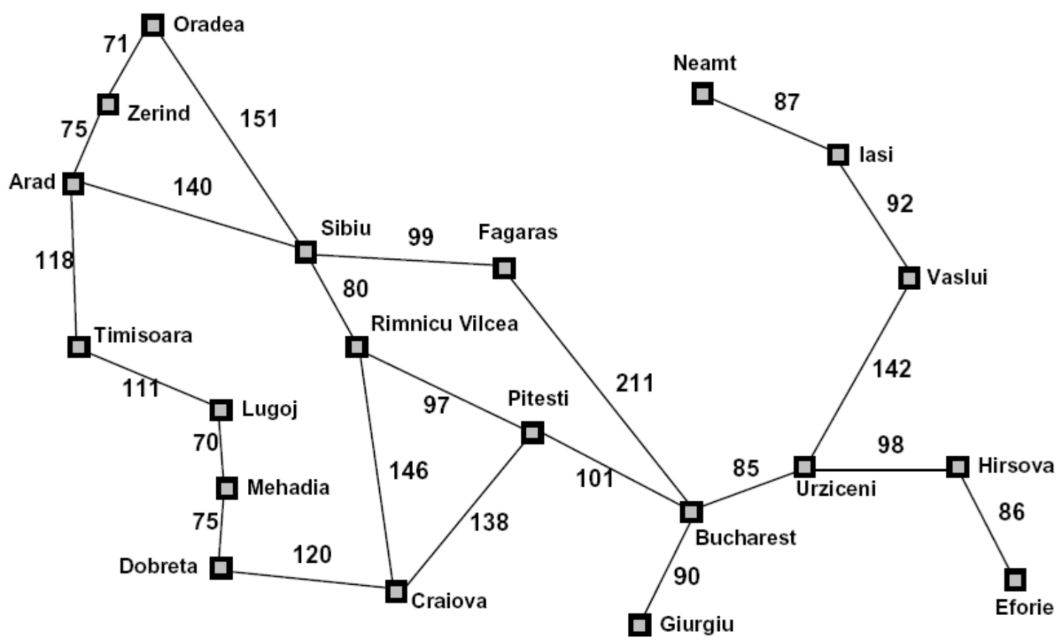
S to n1: → step cost = 10

S to n3: → step cost = 100



Greedy Search

- Completeness
  - Greedy tree search: Incomplete even in a finite state space (does not guarantee to find a solution)



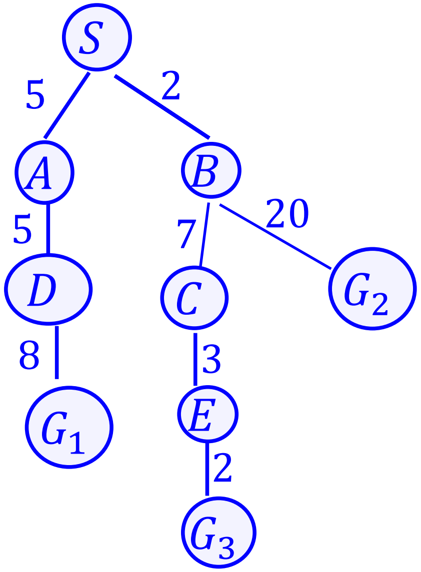
  - The graph search version: Complete in finite state spaces, but not in infinite state spaces

A* Search

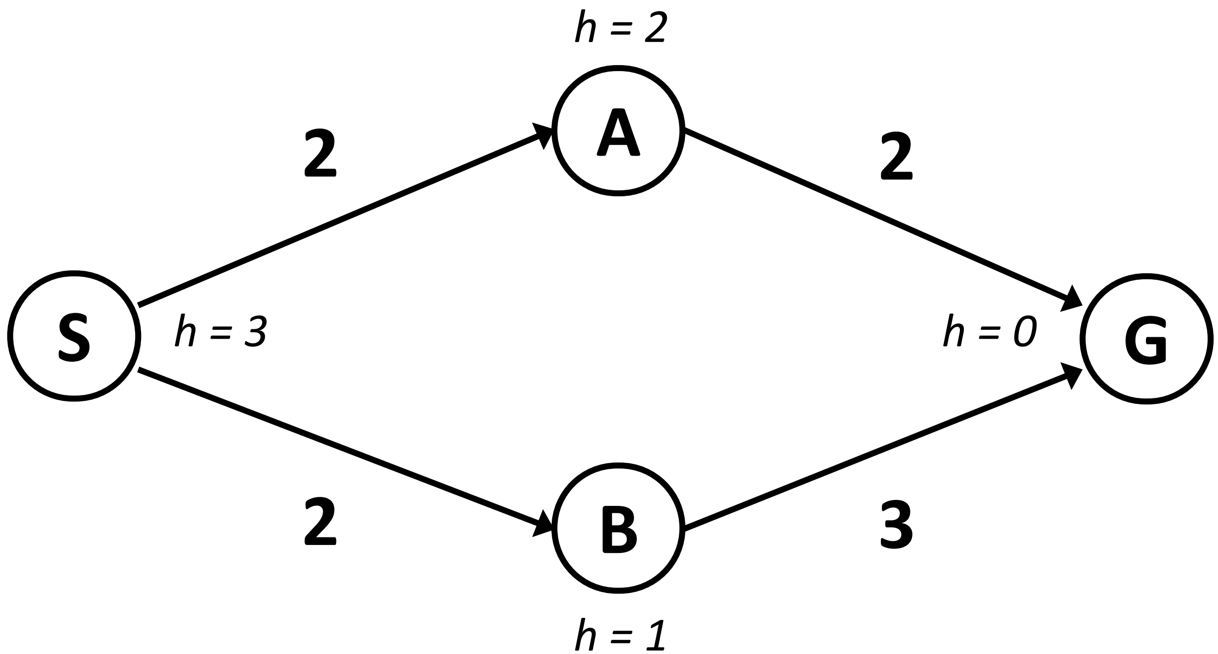
- Minimizes the estimated total cost of a solution path

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

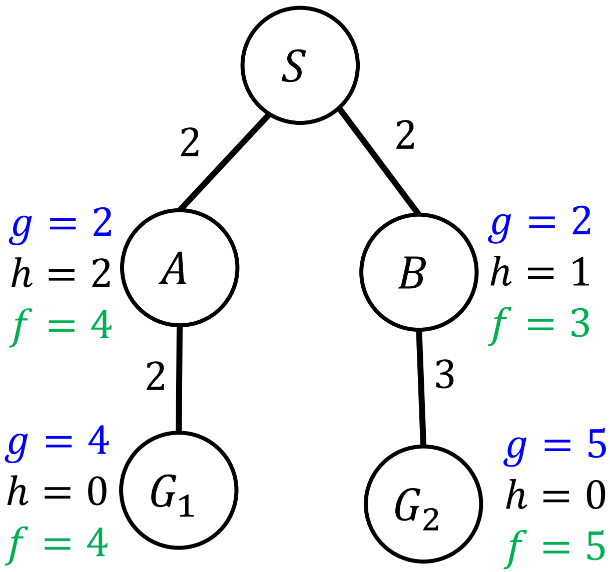
  - $g(n)$ - the path cost from the initial node to node n
  - $h(n)$ - the estimated cheapest cost to get from node n to the goal node
  - $f(n)$ – estimated total cost of the cheapest path that continuous from node n to a goal



A^* Search Example



- Start from S, G is the goal
- Expanded nodes in order: S, B, A, G
- Solution path: S->A->G
- Solution path cost: 4

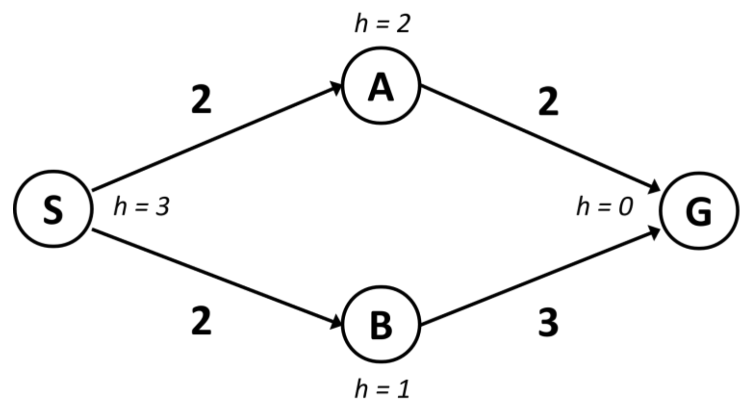


$$A^*: f(n) = g(n) + h(n)$$

Visited	Frontier
S	A, B
B	A, G_2
A	G_2, G_1
G_1	

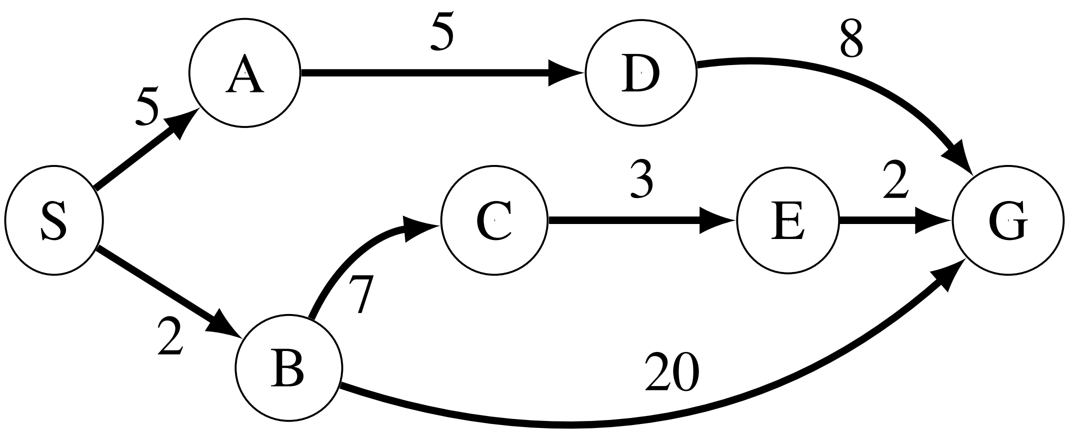
Solution path: $S \rightarrow A \rightarrow G_1$

Solution path cost: 4

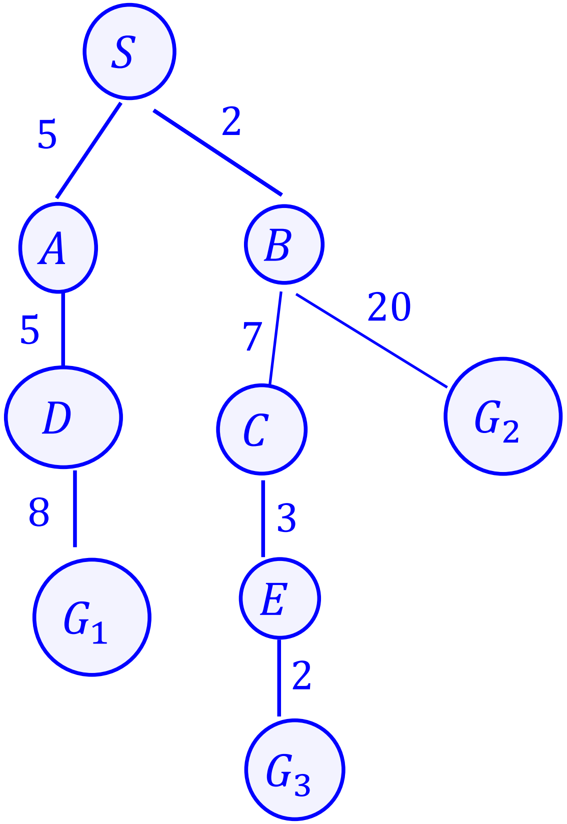


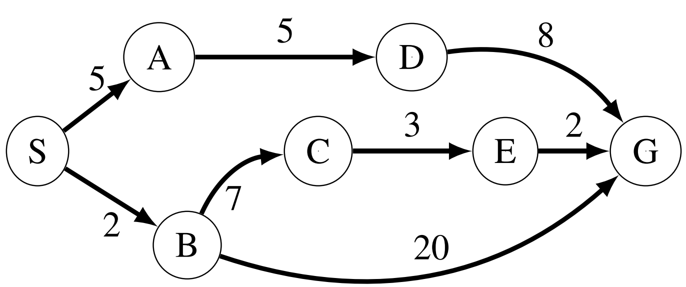
BFS Example

- 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





BFS

Visited:

S, A, B, D, C, G_2

Solution path:

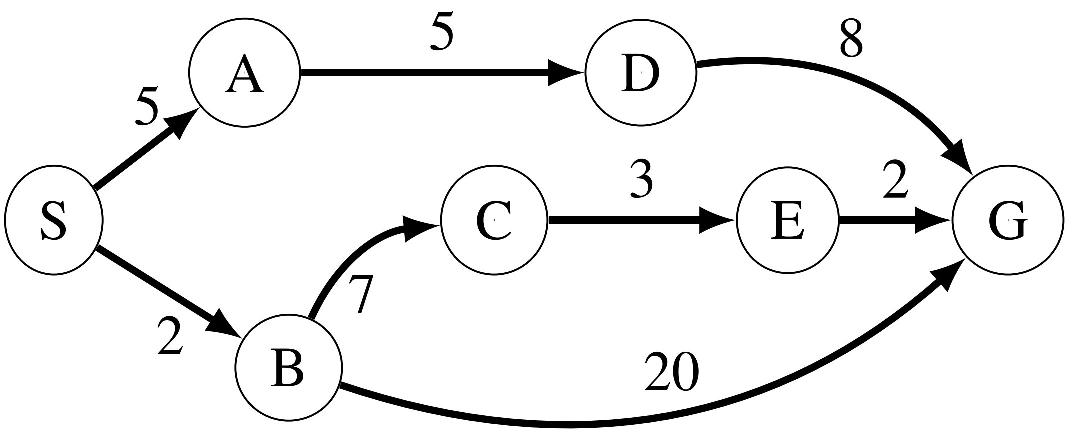
$S \rightarrow B \rightarrow G_2$

Sol. path cost:

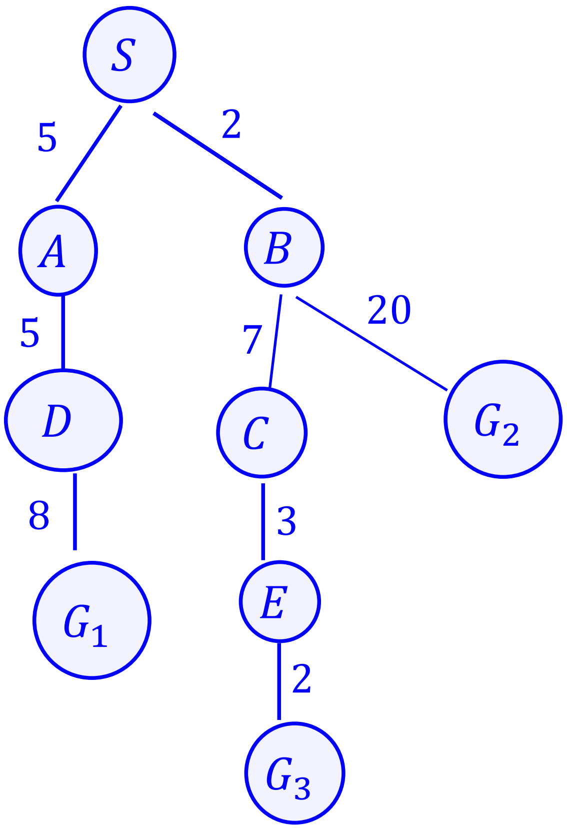
$2 + 20 = 22$

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



DFS

visited:

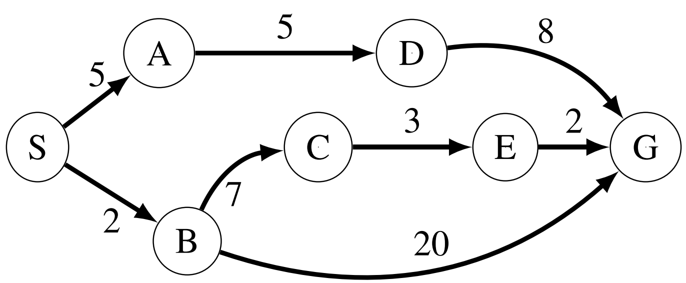
$S \to A \to D \to G_1$

sol. path:

$S \to A \to D \to G_1$

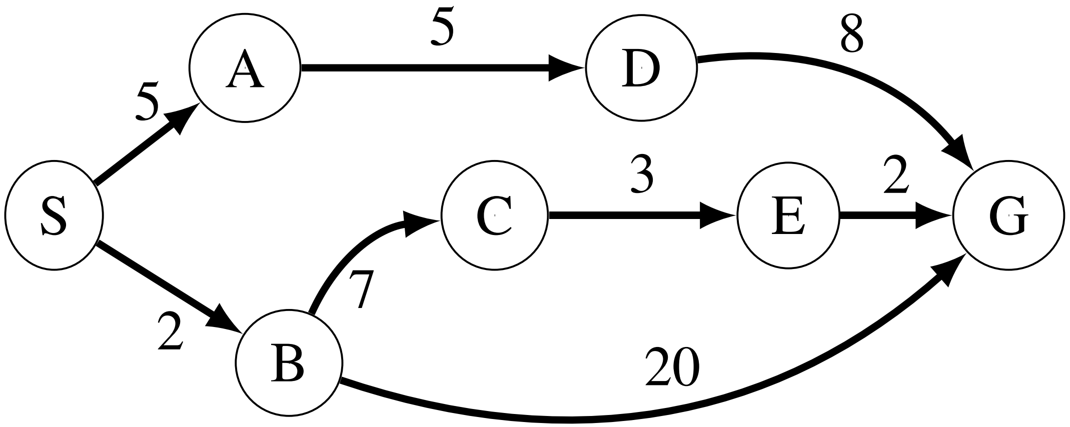
sol. path cost:

$5 + 5 + 8 = 18$

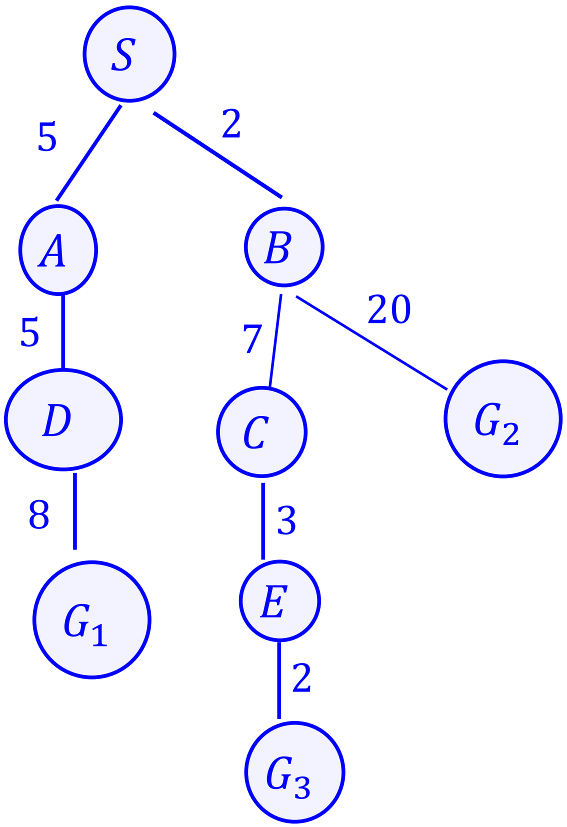


IDDFS Example

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



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



IDDFS

Visited:

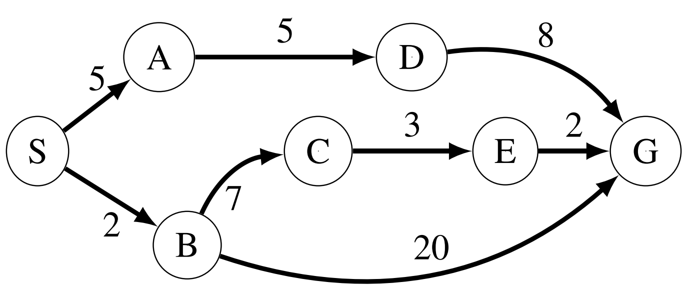
$S;$

$S, A, B;$

S, A, D, B, C, G_2

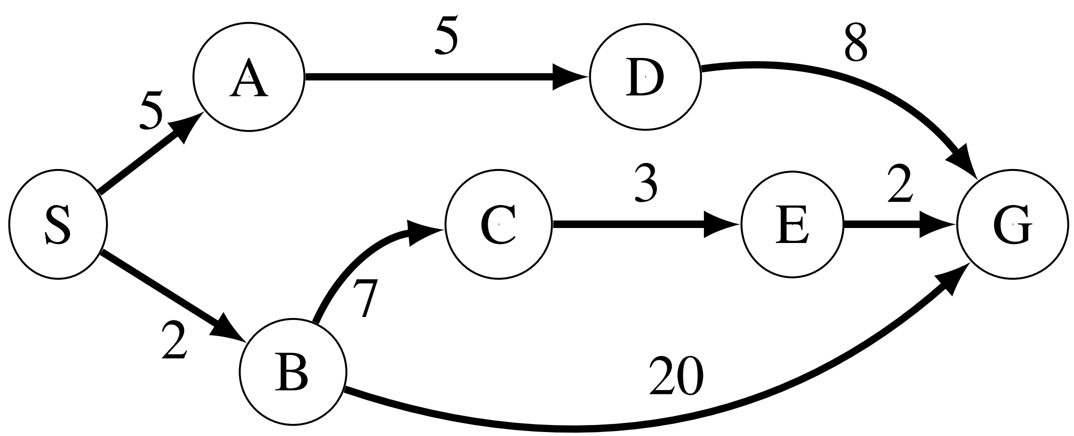
sol. path: $S \rightarrow B \rightarrow G_2$

sol path cost: $2 + 20 = 22$

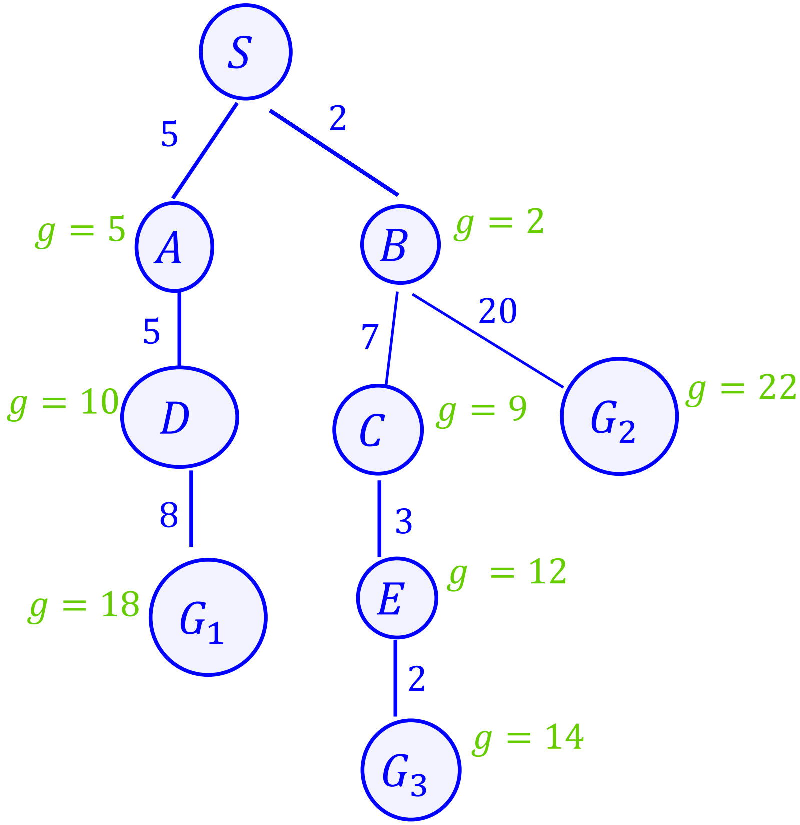


UCS Example

- 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

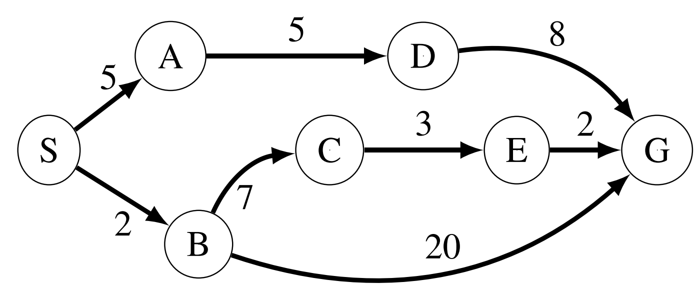


UCS: g valves.

Visited	Frontier
S	A, B
B	A, C, G_2
A	C, G_2, D
C	G_2, D, E
D	G_2, E, G_1
E	G_2, G_1, G_3
G_3	

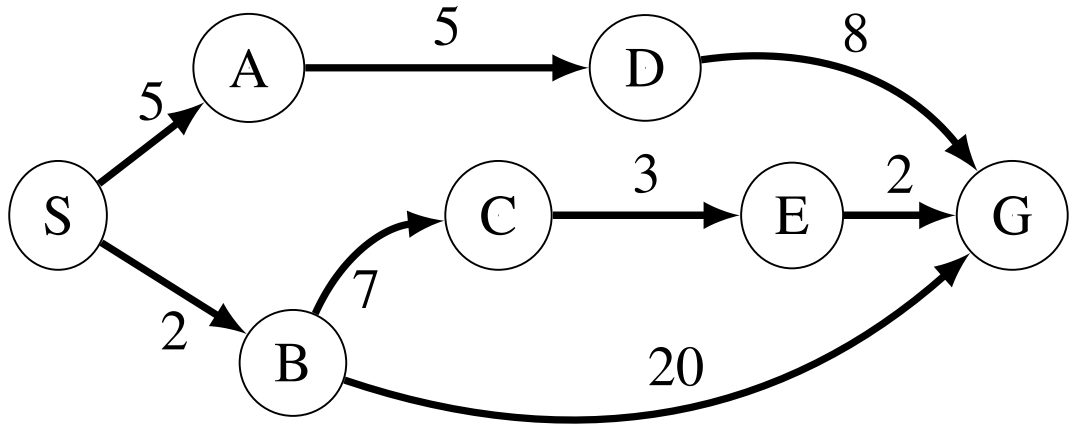
Sol. path: $S \rightarrow B \rightarrow C \rightarrow E \rightarrow G_3$

Sol. path cost: 14



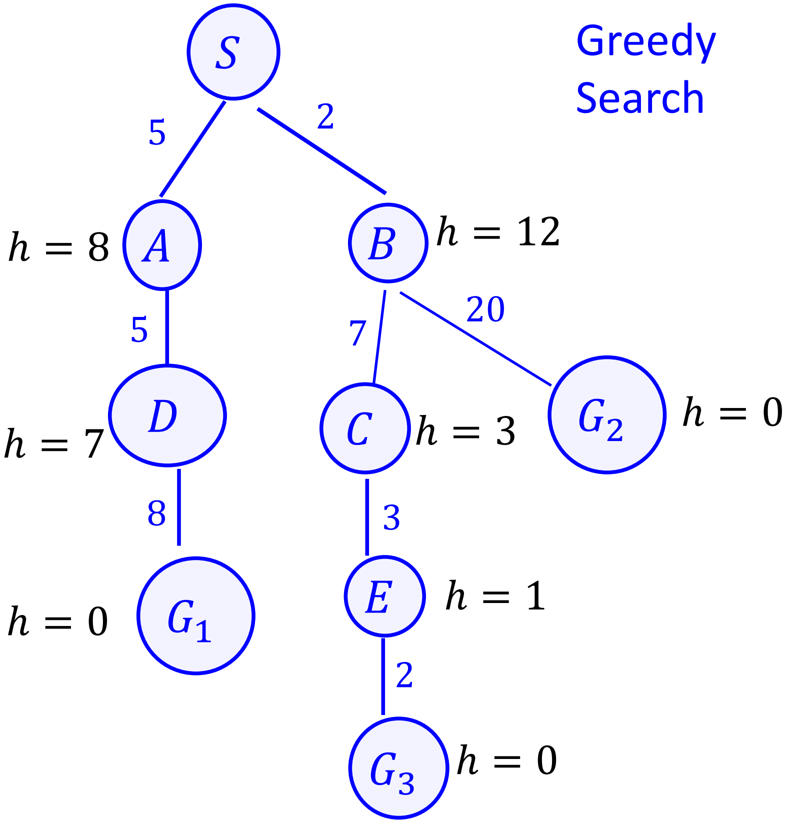
Greedy Search Example

- 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.

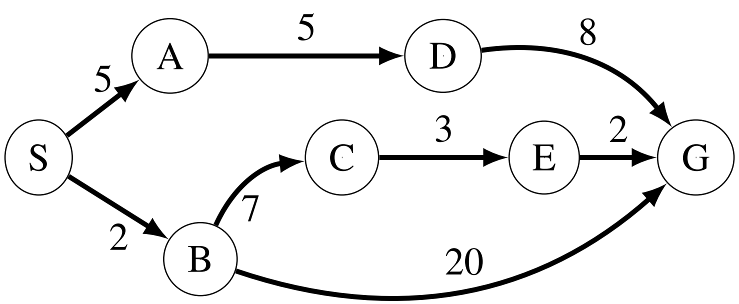


Greedy Search

Visited	Frontier
S	A, B
A	B, D
D	B, G_1
G_1	

solution path: $S \rightarrow A \rightarrow \mathrm{D} \rightarrow G_1$

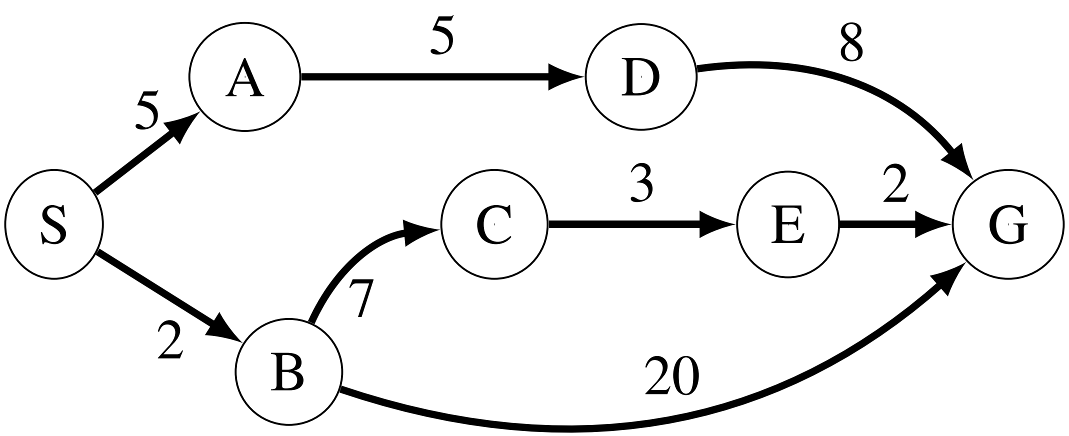
solution path cost: 18



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

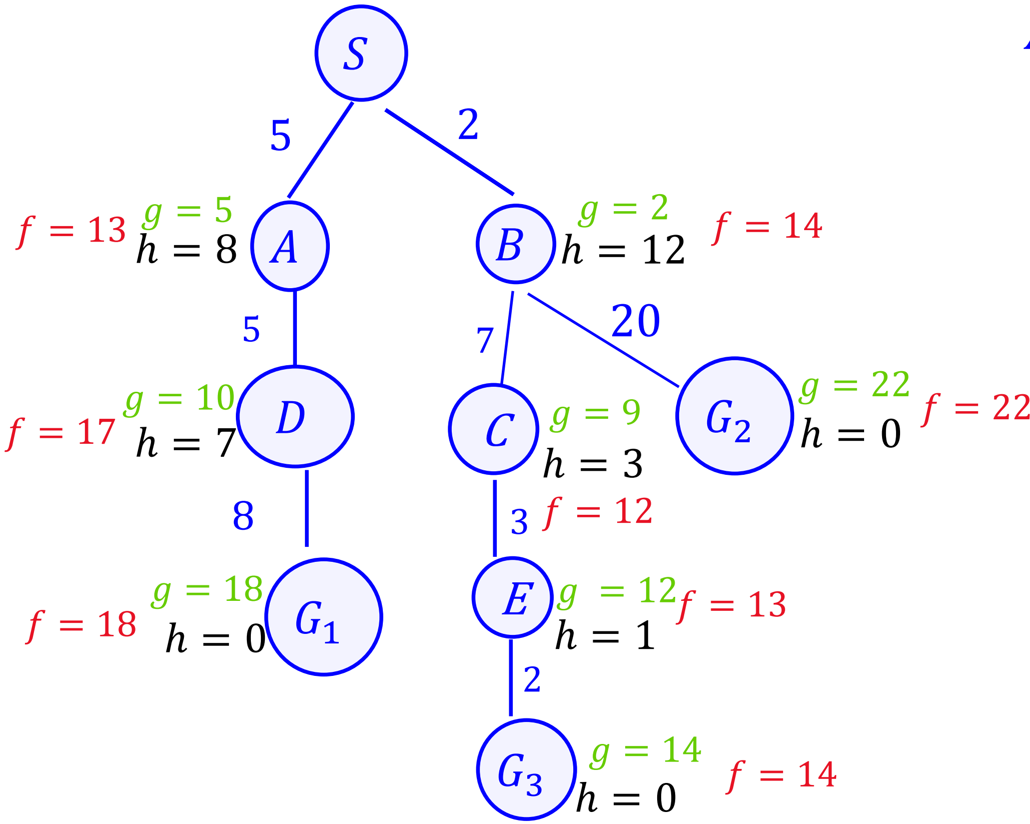
A* Search Example

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



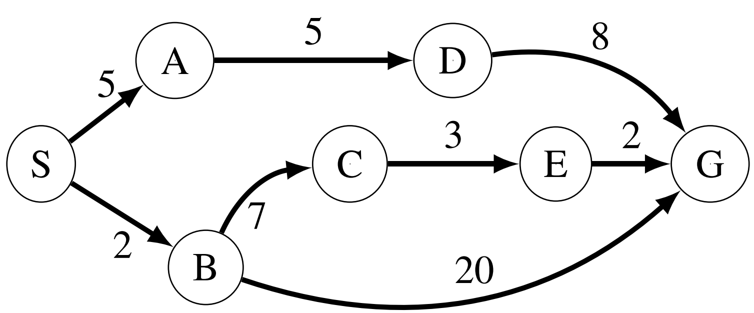
- Answer:
- Expanded nodes: S,A,B,C,E,G
- Solution path: SBCEG
- Path cost: 14

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



$A^*: f(n) = g(n) + h(n)$

Visited	Frontier
S	A, B
A	B, D
B	D, C, G_2
C	D, G_2, E
E	D, G_2, G_3
G_3	



Sol. path: $S \rightarrow B \rightarrow C \rightarrow E \rightarrow G_3$

Sol. path cost: 14

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