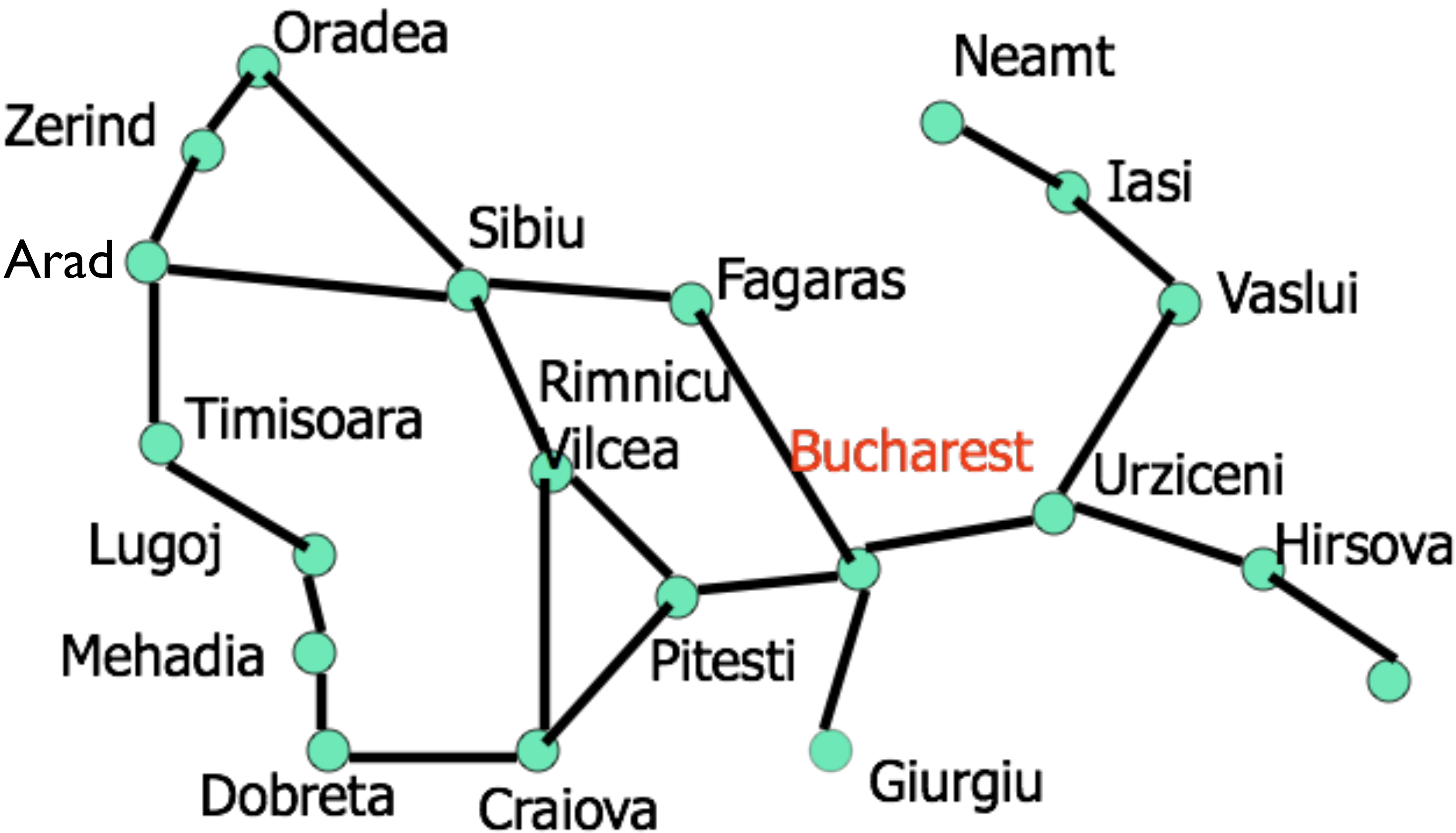
Blind search algorithms: DFS, BFS, iterative deepening

Devika Subramanian

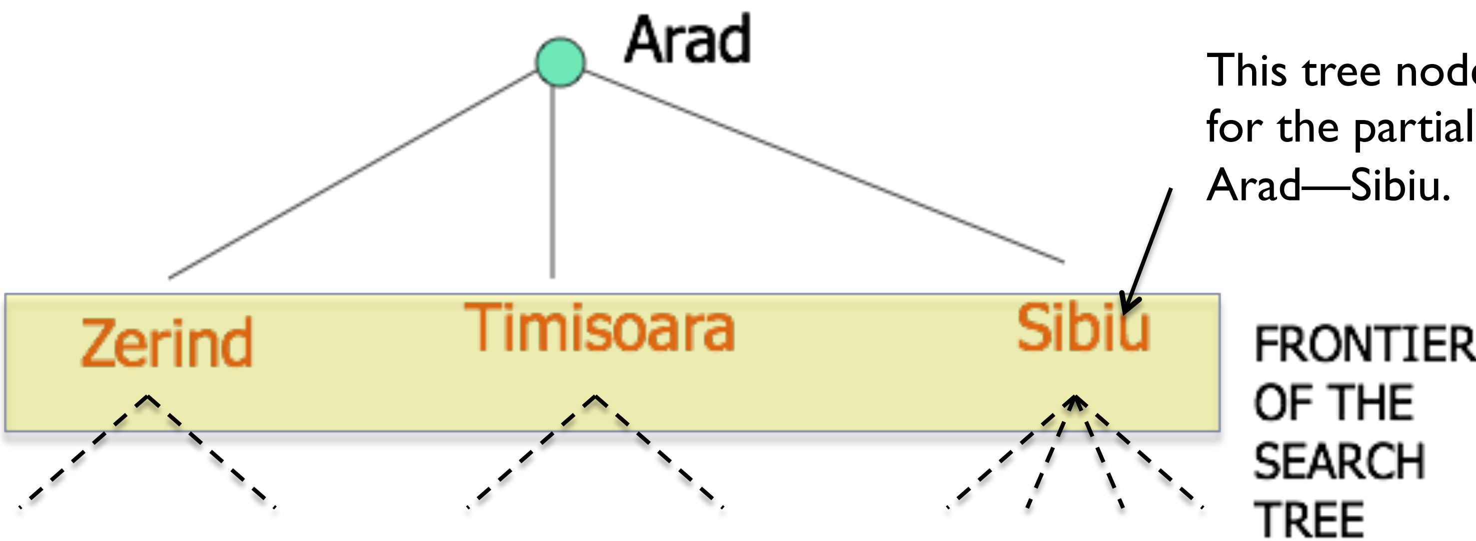
Route planning



Need for search algorithms

- Dynamic programming takes time proportional to the square of the size of the state space.
  - It finds shortest paths to a goal (e.g., Bucharest) from every node in the state space (e.g., every city in Romania).
- What if all we care about is getting between a given node (e.g., Arad) and a goal node (e.g., Bucharest)?
  - Can we solve this problem in time proportional to the size of the state space?
- This is what search algorithms are for: given a start state, a goal state and a state space graph, find a path between the two states.

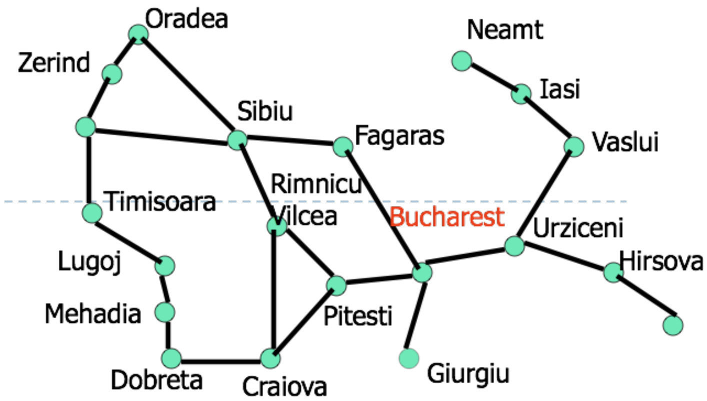
Search tree

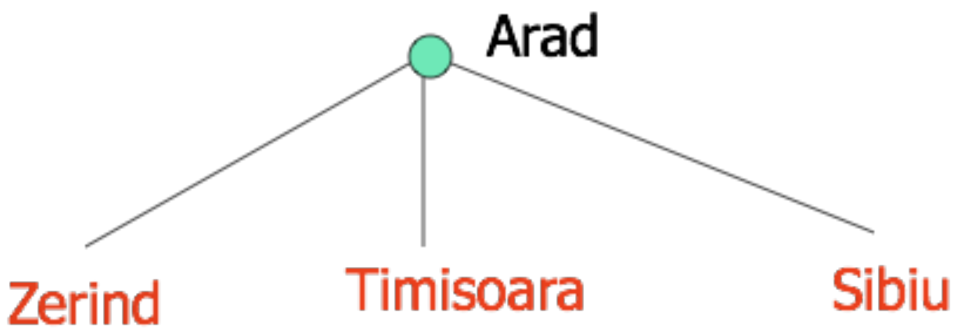


This tree node stands for the partial plan Arad—Sibiu.

A tree is a graph in which any two vertices are connected by exactly one path.

Search tree





FRONTIER OF THE SEARCH TREE

A tree is a graph in which any two vertices are connected by exactly one path.

- A what-if tree of plans and their outcomes
- The root node is the start state
- Children correspond to successor states
- Nodes also correspond to sequences of actions to achieve the state at the node
- For real problems, we never build the entire tree!

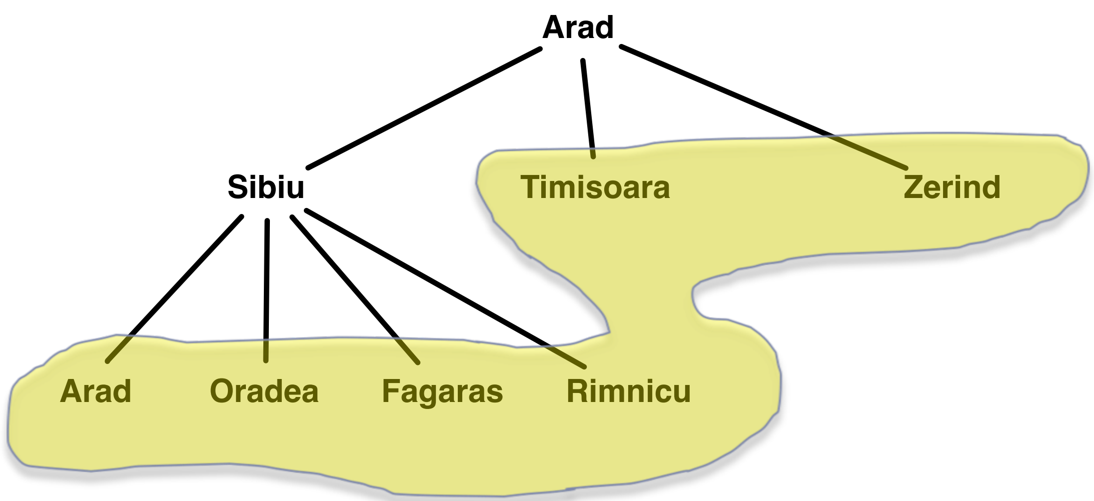
Search algorithms

- Which node in the frontier of the search tree to expand next?
- Two classes of search strategies
  - Uninformed: no information on distance to goal configuration.
  - Informed: use estimates of distance to goal configuration.

Depth-first search

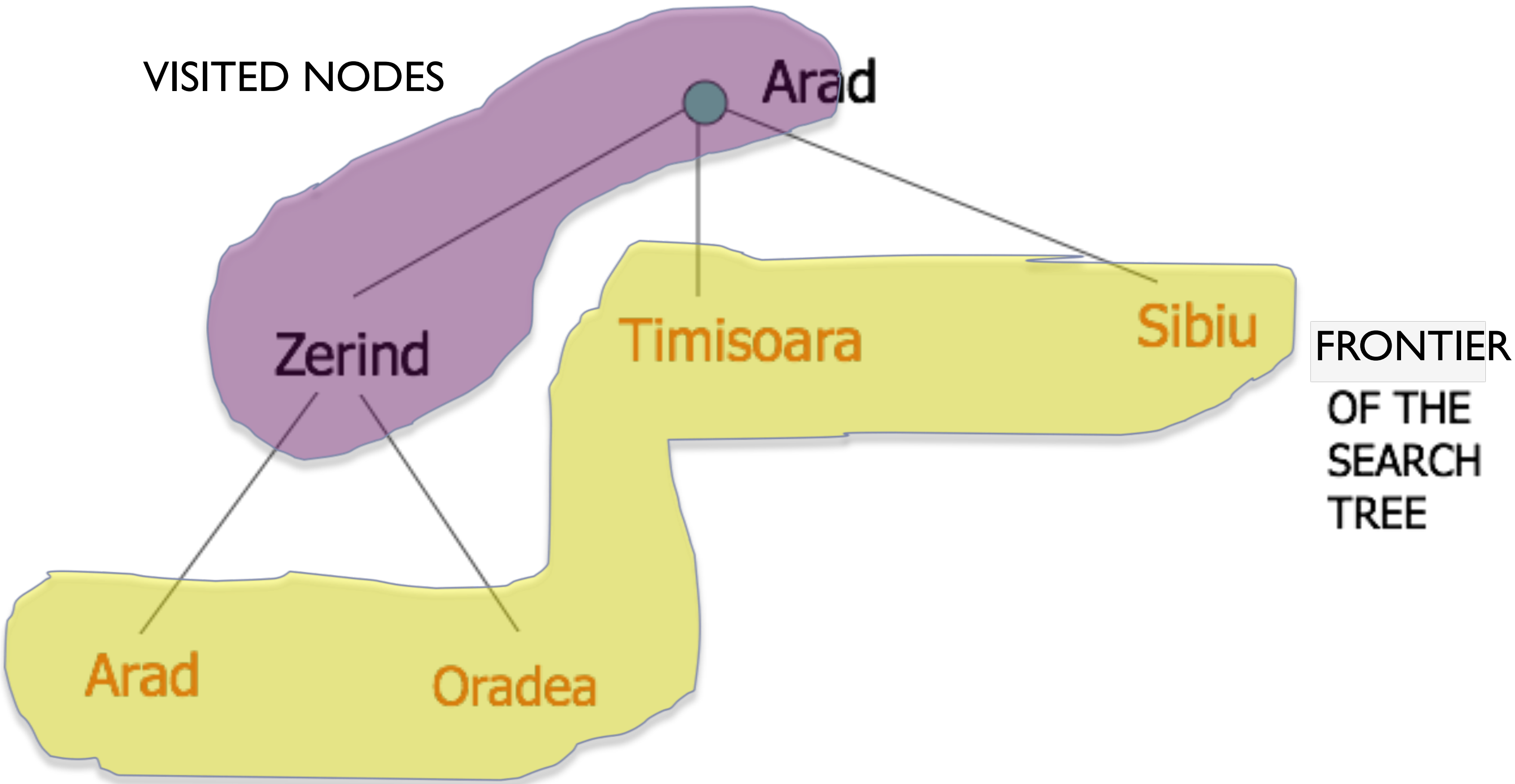
- Always explore from the most recently added node, if it has any untried successors. Else, backup to the previous node on the current path.

Search tree for depth-first search



FRONTIER OF THE SEARCH TREE

Search trees with visited nodes



Template for search algorithms

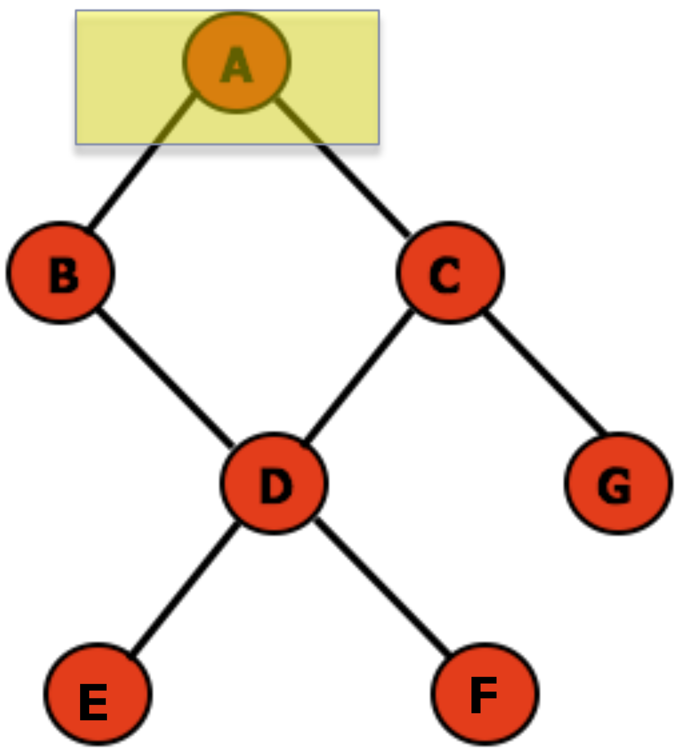
- Function SEARCH(start, end, graph, frontier) returns True or False
  - Insert start into the frontier.
  - Initialize the visited list to empty.
  - while frontier is nonempty:
    - current = **remove** node from frontier
    - add current to the visited list
    - If current node == end, return True.
    - for every nbr of current node not in visited
      - **insert** nbr into frontier
  - return False.

Depth-first search

- Policy for adding and removing entries from the frontier
  - stack: last in, first out

Depth-first search (DFS) : a stack frontier

Find a path from A to G

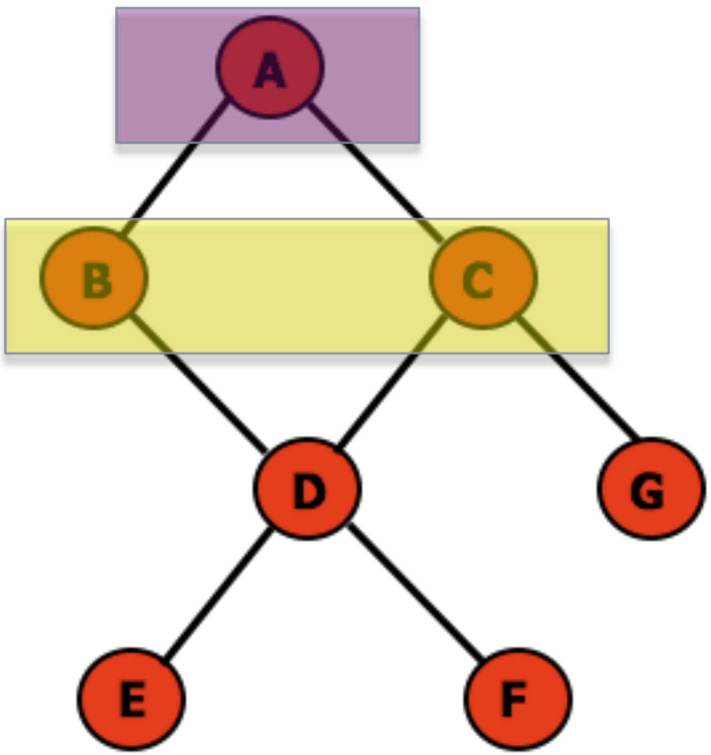


Frontier **Visited**

A

frontier starts with A
visited is empty

Depth-first search (DFS): a stack frontier

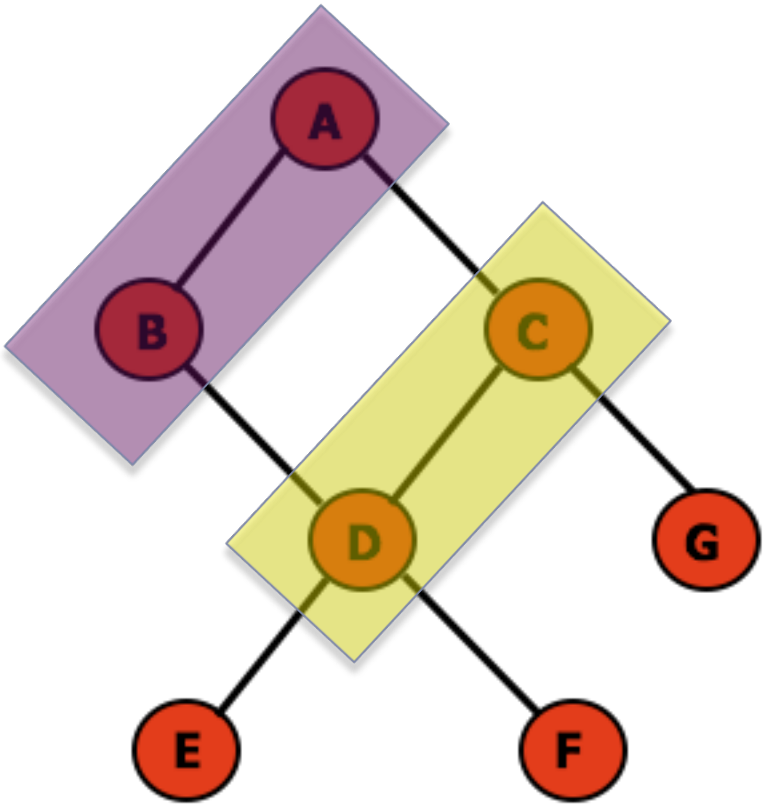


Frontier	Visited
B	A
C	

The top node of frontier is current.
Remove current from frontier
and move it to visited.

for every nbr of current not in visited
  insert nbr into frontier

Depth-first search (DFS): a stack frontier

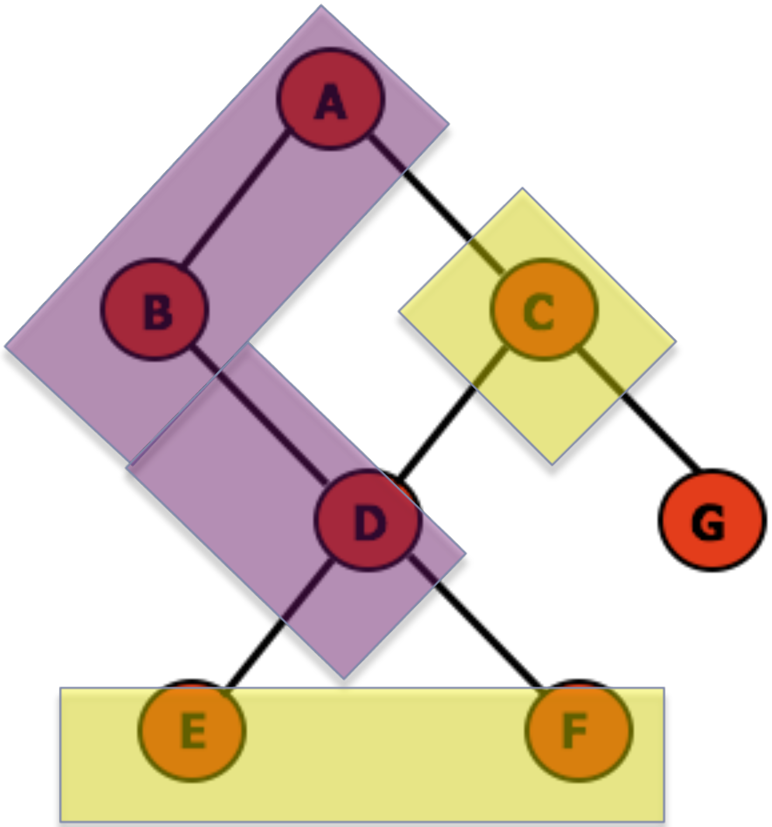


Frontier	Visited
D	A
C	B

The top node of frontier is current.
Remove current from frontier
and move it to visited.

for every nbr of current not in visited
    insert nbr into frontier

Depth-first search (DFS): a stack frontier



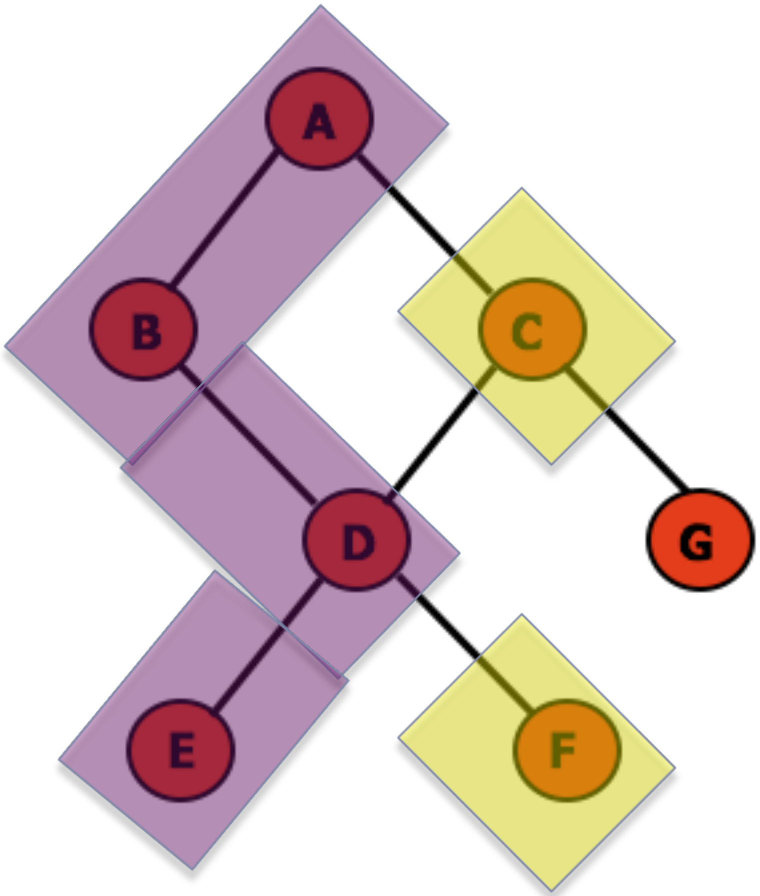
Frontier	Visited
E	A
F	B
C	D
~~C~~	

The top node of frontier is current.
Remove current from frontier
and move it to visited.

for every nbr of current not in visited
   insert nbr into frontier

If you generate a node which is already on the frontier, remove the one already in frontier and insert the new one.

Depth-first search (DFS): a stack frontier

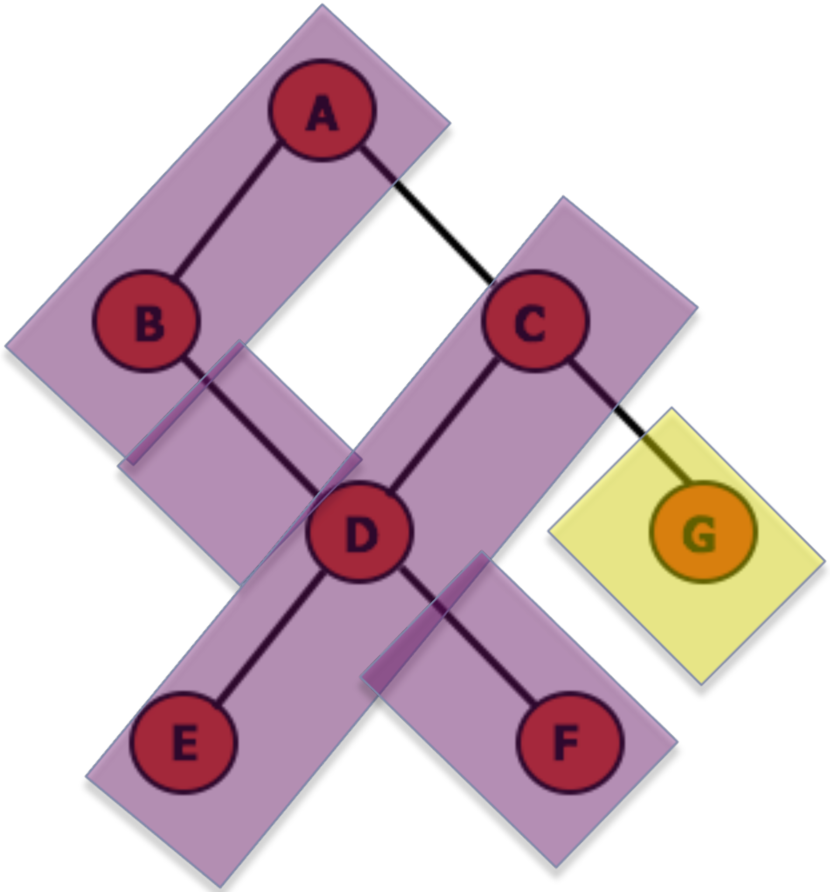


Frontier	Visited
F	A
C	B
	D
	E

The top node of frontier is current.
Remove current from frontier
and move it to visited.

for every nbr of current not in visited
insert nbr into frontier

Depth-first search (DFS): a stack frontier



Frontier	Visited
G	A
	B
	D
	E
	F
	C

The top node of frontier is current.
Remove current from frontier
and move it to visited.

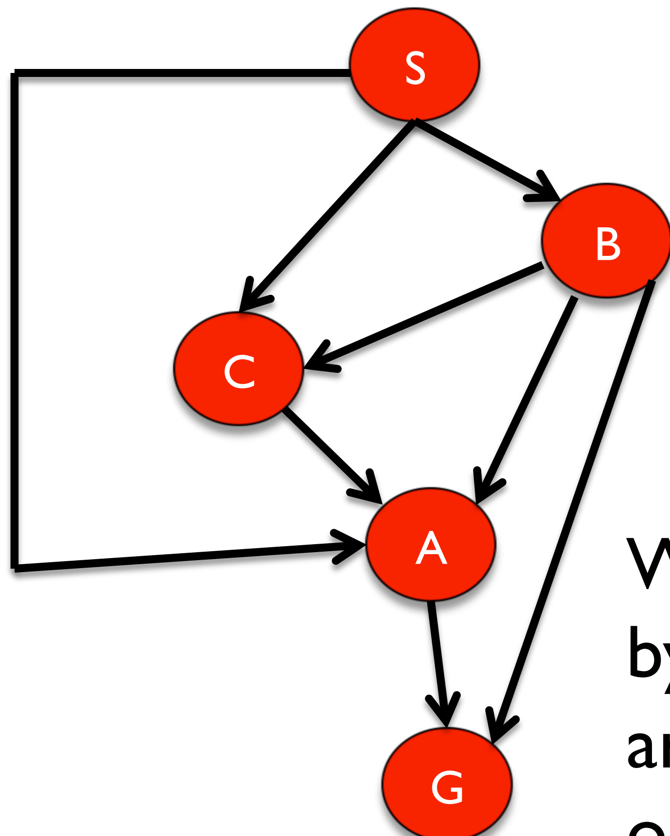
for every nbr of current not in visited
insert nbr into frontier

G is the goal node and it is current. We have a found a path.

How to manage the frontier for DFS

- To get the current node, pop the frontier list
- To insert a new node n,
  - if node n already exists in frontier, remove it from frontier
  - Add the new node n to the front of the frontier list.

Path selected by DFS from S to G



POLL

Which path is selected by DFS assuming ties are broken in alphabetical order?

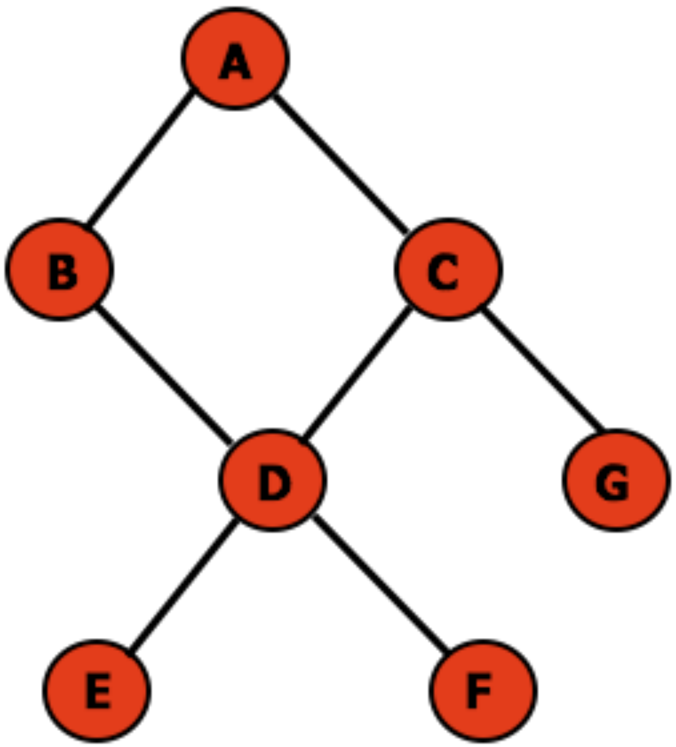
S→ A explored before S→B

Template for search algorithms

- Function SEARCH(start, end, graph, frontier) returns True or False
  - Insert start into the frontier.
  - Initialize the visited list to empty.
  - while frontier is nonempty:
    - current = **remove** node from frontier
    - add current to the visited list
    - If current node == end, return True.
    - for every nbr of current node not in visited
      - **insert** nbr into frontier
  - return False.

Breadth-first search (BFS): a queue frontier

Find a path from A to G

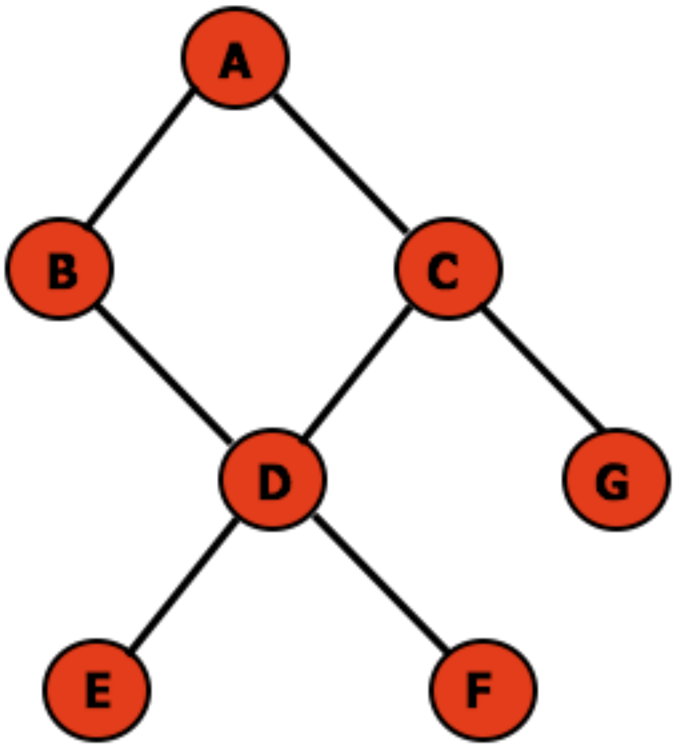


Frontier **Visited**

A

frontier starts with A
visited is empty

Breadth-first search (BFS): queue frontier

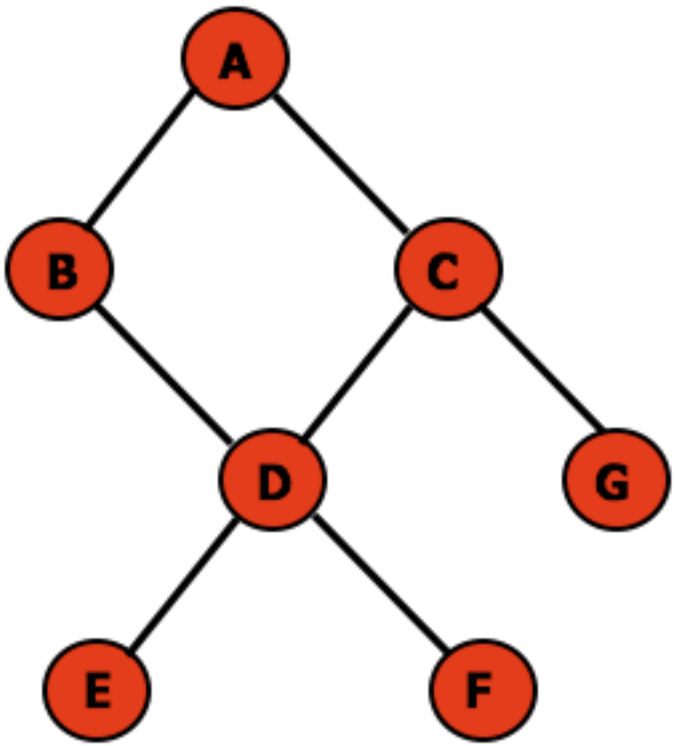


Frontier	Visited
B	A
C	

The top node of frontier is current.
Remove current from frontier
and move it to visited.

for every nbr of current not in visited
  insert nbr into frontier

Breadth-first search (BFS): queue frontier

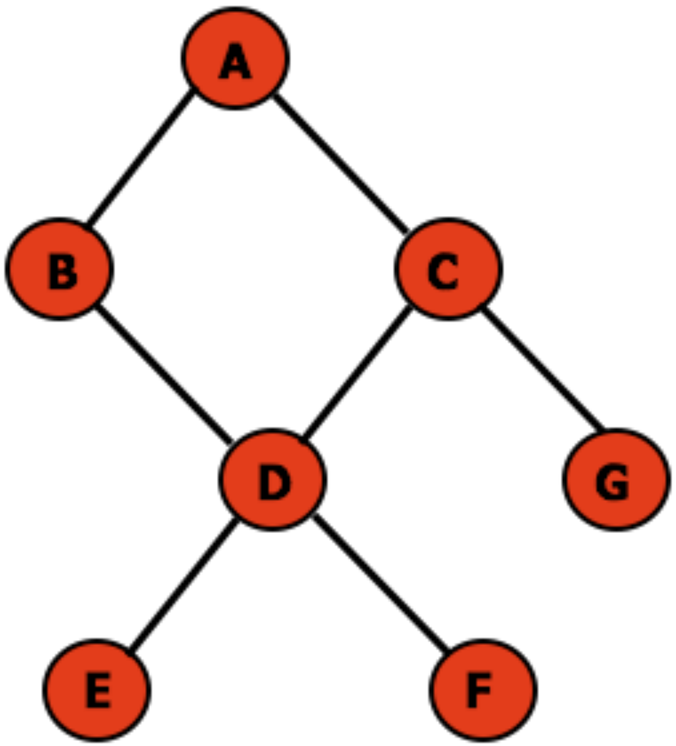


Frontier	Visited
C	A
D	B

The top node of frontier is current.
Remove current from frontier
and move it to visited.

for every nbr of current not in visited
    insert nbr into frontier

Breadth-first search (BFS): queue frontier



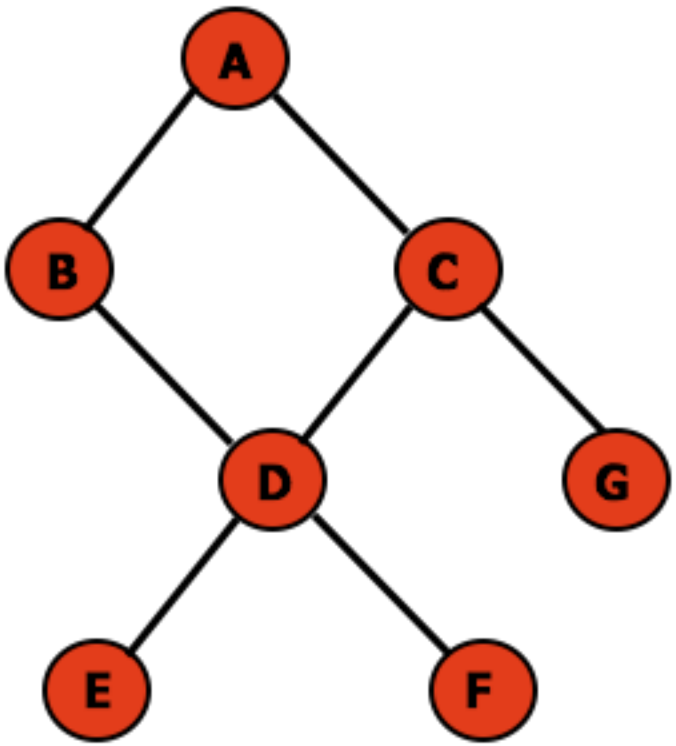
Frontier	Visited
D	A
G	B
	C

The top node of frontier is current.
Remove current from frontier
and move it to visited.

for every nbr of current not in visited
  insert nbr into frontier

If you generate a node which is already on the frontier, do not insert it into frontier.

Breadth-first search (BFS): queue frontier

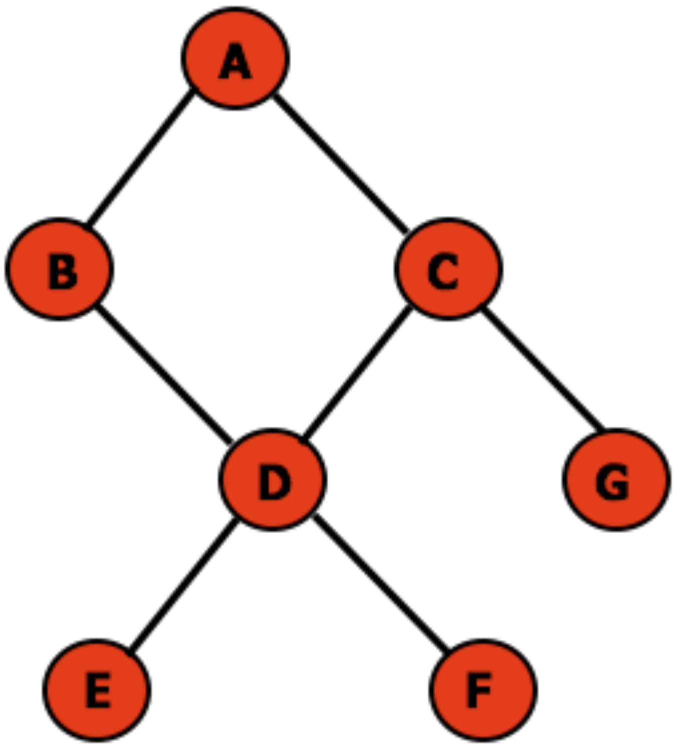


Frontier	Visited
G	A
E	B
F	C
	D

The top node of frontier is current.
Remove current from frontier
and move it to visited.

for every nbr of current not in visited
    insert nbr into frontier

Breadth-first search (BFS): queue frontier



Frontier	Visited
G	A
E	B
F	C
	D

The top node of frontier is current.
Remove current from frontier
and move it to visited.

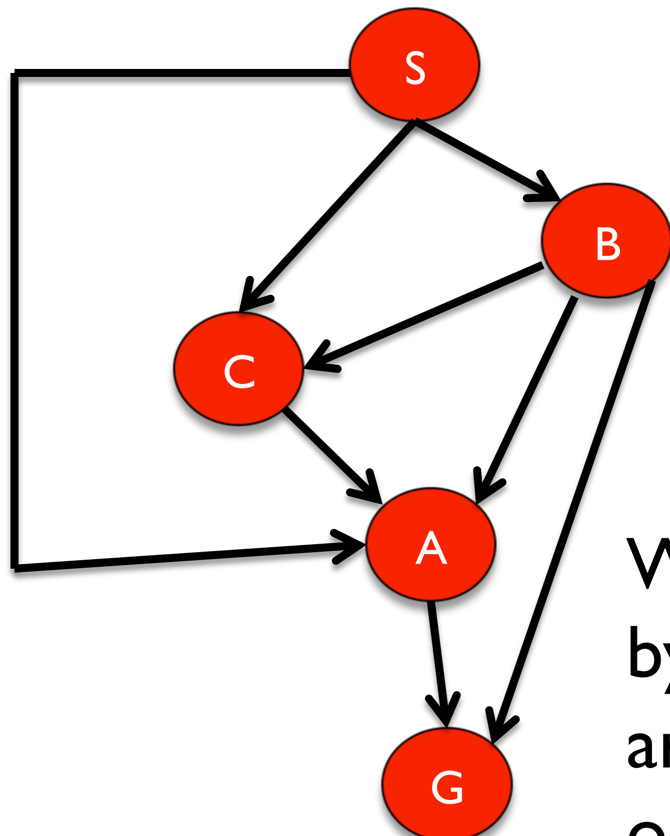
for every nbr of current not in visited
  insert nbr into frontier

G is the goal node and it is current. We have a found a path.

How to manage the frontier for BFS

- To get the current node, get the node at the front of the frontier list
- To insert a new node n,
  - if node n already exists in frontier, do not insert it into frontier
  - Add the new node n to the back of the frontier list.

Path selected by BFS from S to G



POLL

Which path is selected by BFS assuming ties are broken in alphabetical order?

S→ A explored before S→B

Template for search algorithms

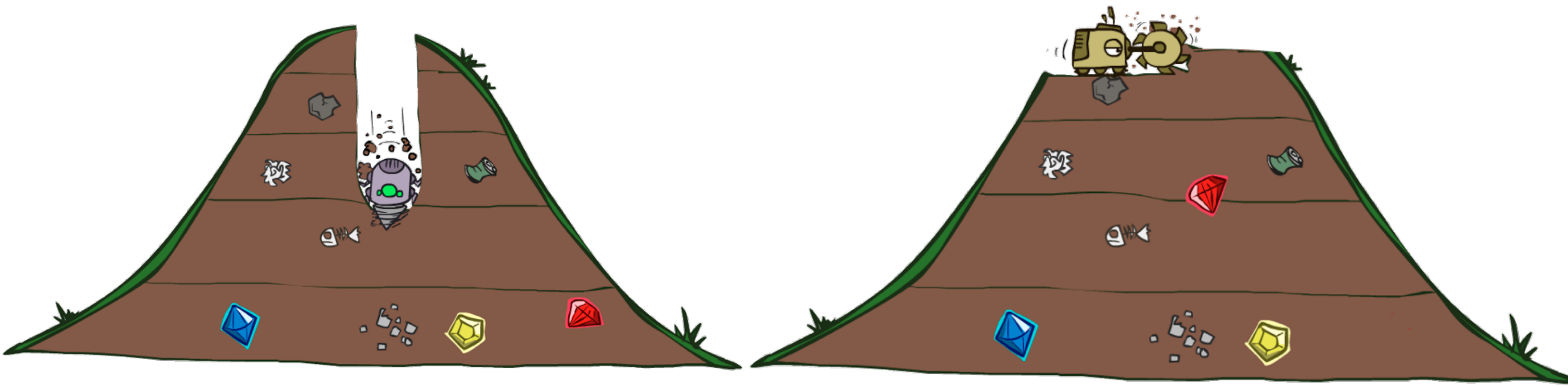
- Function SEARCH(start, end, graph, frontier) returns True or False
  - Insert start into the frontier.
  - Initialize the visited list to empty.
  - while frontier is nonempty:
    - current = **remove** node from frontier
    - add current to the visited list
    - If current node == end, return True.
    - for every nbr of current node not in visited
      - **insert** nbr into frontier
  - return False.

REMOVAL from frontier:
Last-in first-out: DFS
First-in first-out: BFS

INSERTION into frontier:
Front of list: DFS
End of list: BFS

DUPLICATES in frontier:
Replace with new: DFS
Delete new: BFS

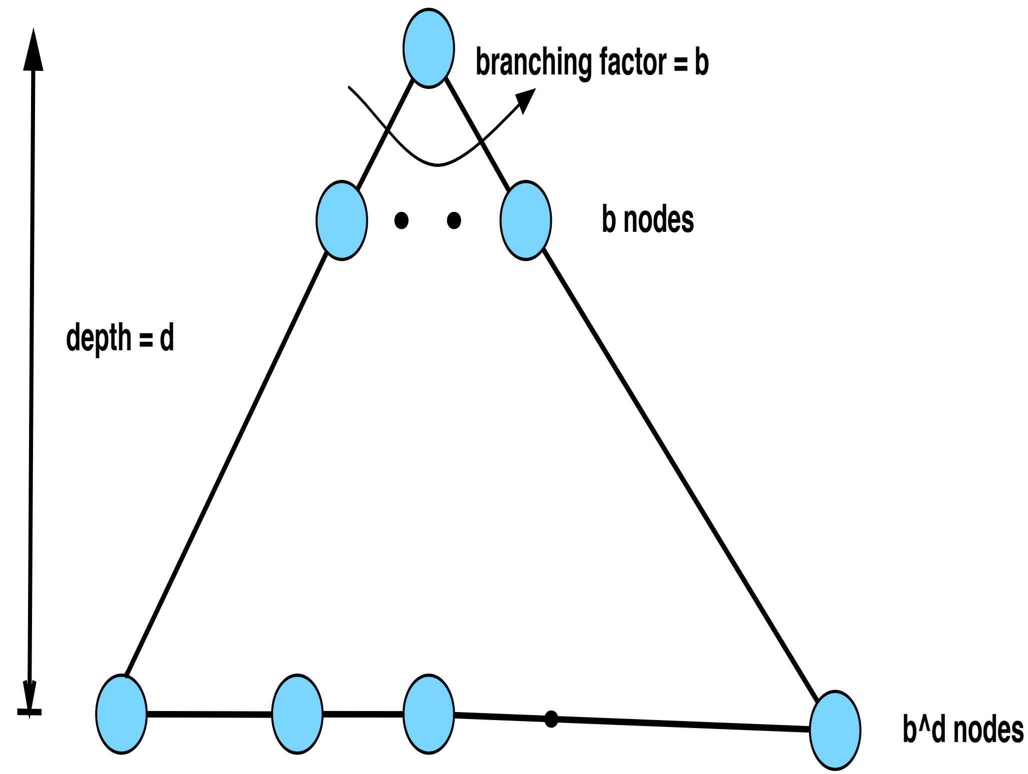
DFS vs BFS



Slide courtesy of Dan Klein/Peter Abeel

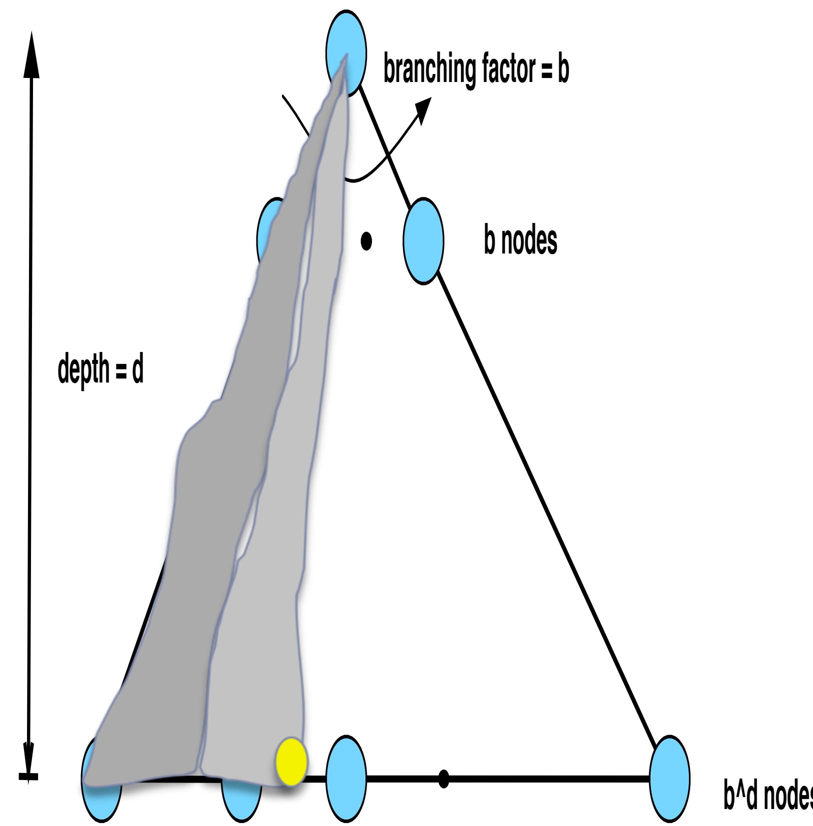
Search Algorithm properties

- Complete: guaranteed to find a solution if one exists.
- Optimal: Guaranteed to find the least cost path.
- Time complexity: big-O analysis in terms of branching factor b and depth d (i.e., size of the state space)
- Space complexity: big-O analysis in terms of branching factor b and depth d (i.e., size of the state space)



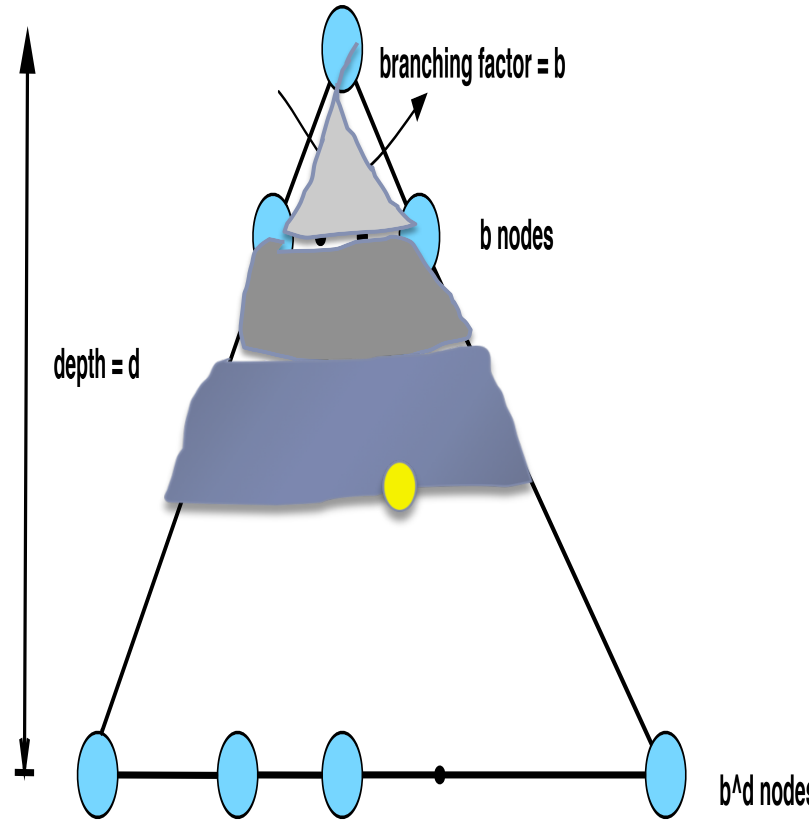
Properties of DFS

- Is DFS complete?
  - If we can prevent cycles, DFS is complete.
- Is DFS optimal?
  - No, it finds the "leftmost" solution.
- Time complexity
  - If no infinite paths, then $O(b^d)$
- Space complexity
  - Only has children on path to root, so O(bd)



Properties of BFS

- Is BFS complete?
  - If there is a solution, BFS will find it.
- Is BFS optimal?
  - Yes, if all edge costs are 1; no, otherwise because BFS finds "shallowest" solution.
- Time complexity
  - $O(b^s)$ if s is the depth of the solution.
- Space complexity
  - $O(b^s)$ is s is the depth of the solution.



DFS v BFS

- Give an example where BFS outperforms DFS.
- Give an example where DFS outperforms BFS.

Depth-limited DFS

- Function depth-limited-DFS (start, end, graph, frontier,d) returns True or False
  - Insert start into the stack frontier.
  - Initialize the visited list to empty.
  - while frontier is nonempty:
    - current = **remove** node from frontier
    - add current to the visited list
    - If current node == end, return True.
    - for every nbr of current node not in visited
      - **insert** nbr into frontier if depth of nbr <= d
  - return False.

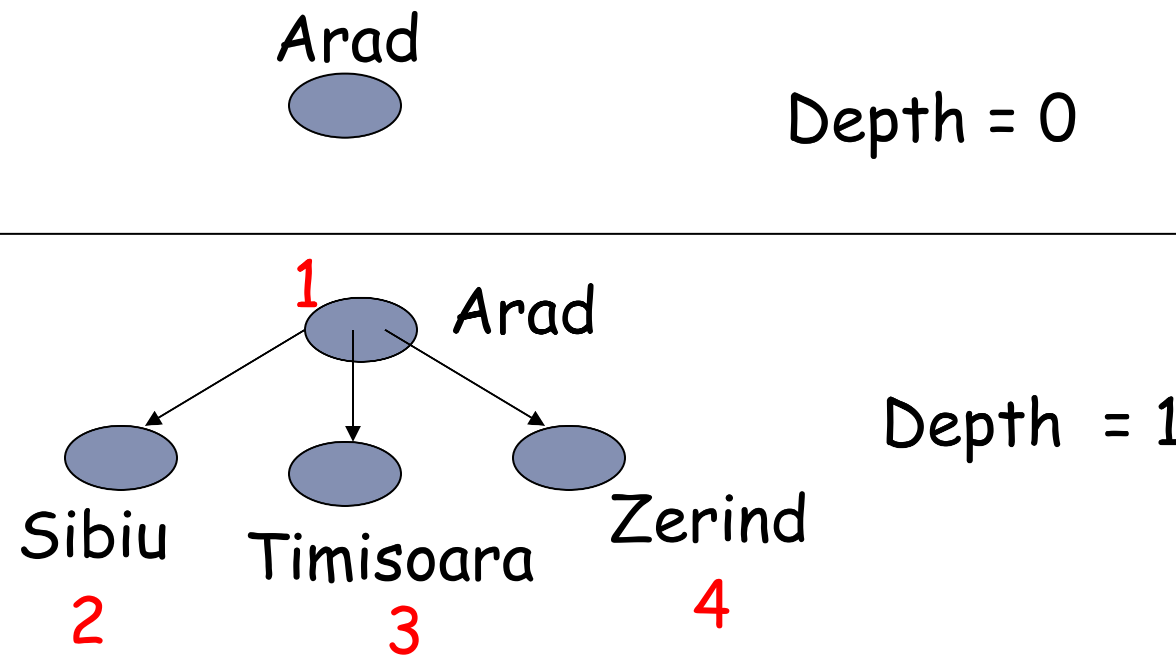
Depth-limited DFS properties

- Is it complete?
- Is it optimal?
- What is its time complexity?
- What is its space complexity?

Iterative deepening

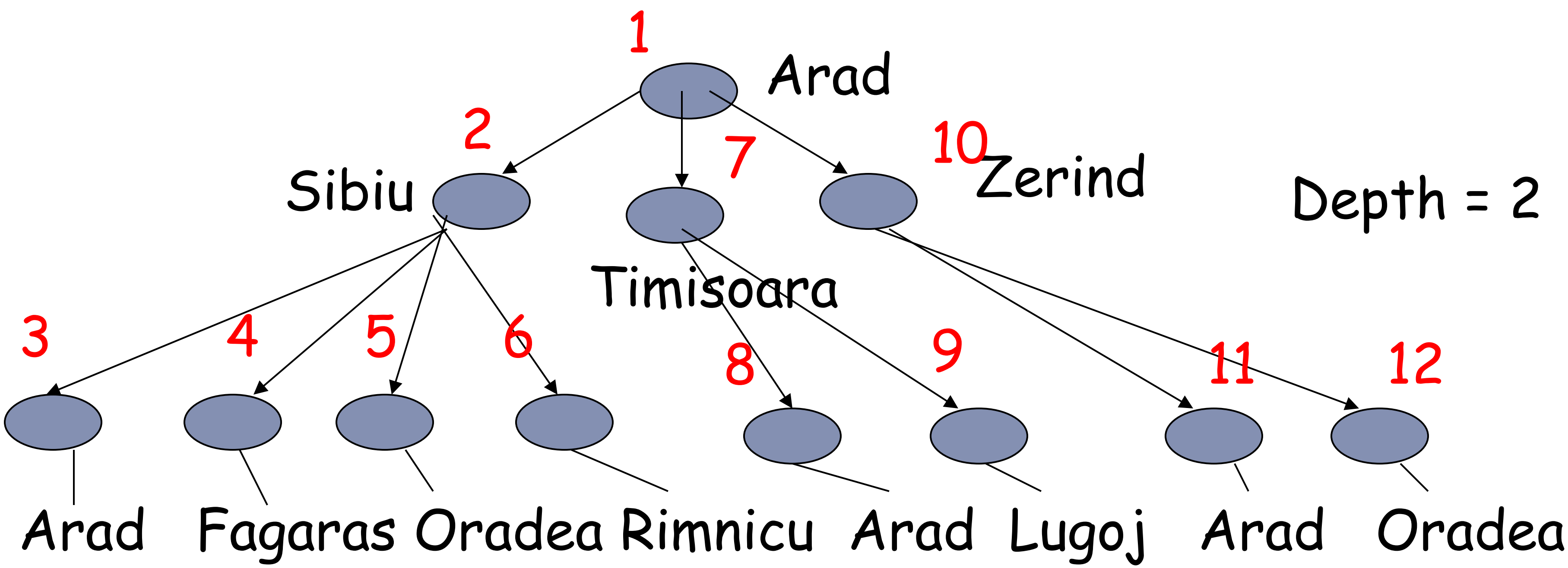
- Function iterative-deepening (start,end,graph,frontier) returns a solution or failure
  - for depth = 0 to ∞ do
    - if depth-limited-DFS(start,end,graph,frontier,d) succeeds then return its result
  - end for
  - return failure

Iterative deepening in action



Tree generated in a depth-first fashion. Order of expansion of nodes indicated in red.

Iterative deepening in action (contd.)



Order of node expansions indicated in red.

Properties of iterative deepening search

- It is complete. It systematically considers all paths of lengths 1,2,3, ……
- It is optimal when all edge costs are 1; it finds the shortest-hop solution. Not optimal for general edge costs.
- Its time complexity is $O(b^d)$, where b is the branching factor and d, the solution depth of the search space.
  - $O(b) + O(b^2) + \ldots + O(b^d) = O(b^d)$
- Its space complexity is $O(bd)$.