CS 1571 Introduction to AI
Lecture 4

Uninformed search methods I.

Milos Hauskrecht
milos@cs.pitt.edu
5329 Sennott Square

Problem-solving as search

- Many search problems can be converted to graph search problems
- **A graph search problem can be described in terms of:**
  - **A set of states** representing different world situations
  - **Initial state**
  - **Goal condition**
  - **Operators** defining valid moves between states
- **Two types of search:**
  - **Path search:** solution **is a path** to a goal state
  - **Configuration search:** solution is **a state** satisfying the goal condition
- **Optimal solution** = a solution with the optimal value
  - shortest path between the two cities, or
  - a desired n-queen configuration

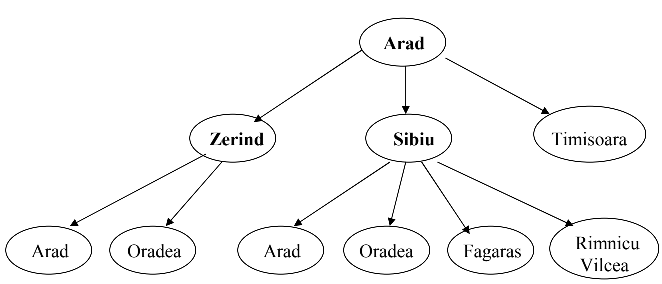
Formulating a search problem

- **Search (process)**
  – The process of exploration of the search space
- **The efficiency of the search depends on:**
  – The search space and its size
  – Method used to explore (traverse) the search space
  – Condition to test the satisfaction of the search objective (what it takes to determine I found the desired goal object)

- **Think twice before solving the problem by search:**
  – Choose the **search space** and the **exploration policy**

Search process

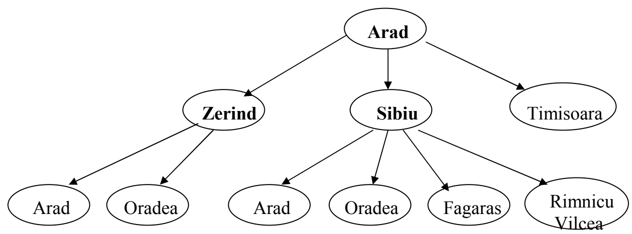
- Exploration of the state space through successive application of operators from the initial state
- A **search tree** = a kind of (search) exploration trace, branches corresponding to explored paths, and leaf nodes corresponding to exploration fringe



Search tree

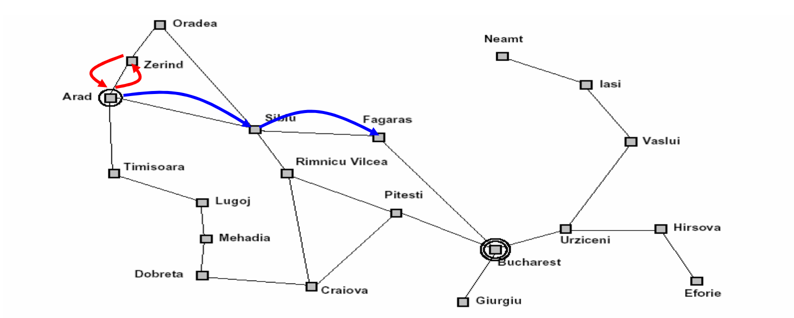
- A **search tree** = a (search) exploration trace
  - **It is different from the graph defining the problem**
  - States can repeat in the search tree

Graph

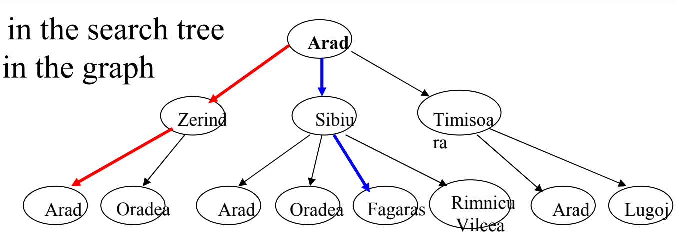


Search tree

Search tree



A branch in the search tree
= path in the graph

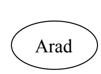


General search algorithm

General-search (*problem, strategy*)
initialize the search tree with the initial state of *problem*
loop
  if there are no candidate states to explore **return** failure
  choose a leaf node of the tree to expand next according to *strategy*
  if the node satisfies the goal condition **return** the solution
  expand the node and add all of its successors to the tree
end loop

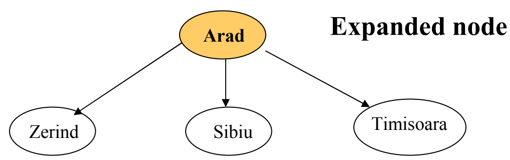
General search algorithm

General-search (*problem, strategy*)
initialize the search tree with the initial state of *problem*
loop
  if there are no candidate states to explore **return** failure
  choose a leaf node of the tree to expand next according to *strategy*
  if the node satisfies the goal condition **return** the solution
  expand the node and add all of its successors to the tree
end loop



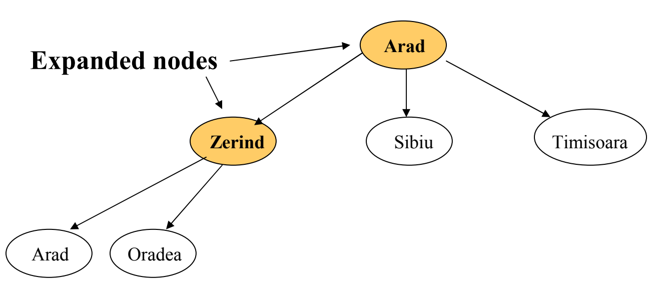
General search algorithm

General-search (*problem, strategy*)
initialize the search tree with the initial state of *problem*
loop
  if there are no candidate states to explore **return** failure
  choose a leaf node of the tree to expand next according to *strategy*
  if the node satisfies the goal condition **return** the solution
  expand the node and add all of its successors to the tree
end loop



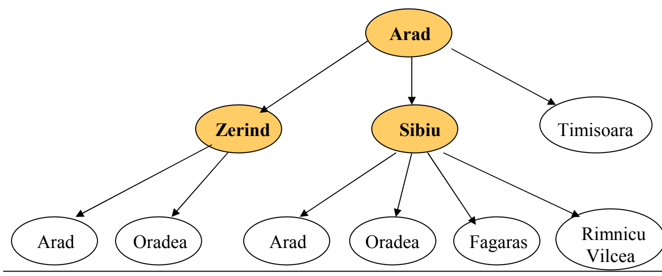
General search algorithm

General-search (*problem, strategy*)
initialize the search tree with the initial state of *problem*
loop
  if there are no candidate states to explore **return** failure
  choose a leaf node of the tree to expand next according to *strategy*
  if the node satisfies the goal condition **return** the solution
  expand the node and add all of its successors to the tree
end loop



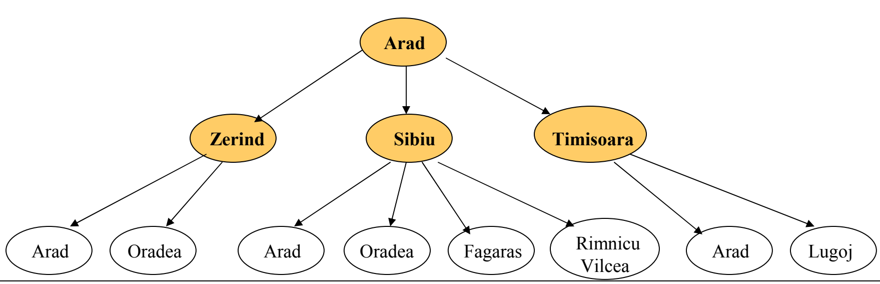
General search algorithm

General-search (*problem, strategy*)
initialize the search tree with the initial state of *problem*
loop
  if there are no candidate states to explore **return** failure
  choose a leaf node of the tree to expand next according to *strategy*
  if the node satisfies the goal condition **return** the solution
  expand the node and add all of its successors to the tree
end loop



General search algorithm

General-search (*problem, strategy*)
initialize the search tree with the initial state of *problem*
loop
  if there are no candidate states to explore **return** failure
  choose a leaf node of the tree to expand next according to *strategy*
  if the node satisfies the goal condition **return** the solution
  expand the node and add all of its successors to the tree
end loop



General search algorithm

General-search (*problem, strategy*)
initialize the search tree with the initial state of *problem*
loop
  if there are no candidate states to explore **return** failure
  choose a leaf node of the tree to expand next according to *strategy*
  if the node satisfies the goal condition **return** the solution
  expand the node and add all of its successors to the tree
end loop

- **Search methods differ in how they explore the space, that is how they choose the node to expand next !!!!!**

Implementation of search

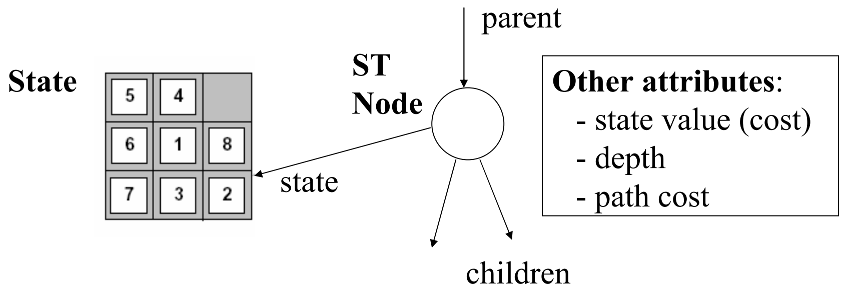
- Search methods can be implemented using **queue** structure

General search (*problem*, Queuing-fn)
  nodes $\leftarrow$ Make-queue(Make-node(Initial-state(*problem*)))
  loop
    if nodes is empty **then return** failure
    node $\leftarrow$ Remove-node(nodes)
    if Goal-test(*problem*) applied to State(*node*) is satisfied **then return** node
    nodes $\leftarrow$ Queuing-fn(nodes, Expand(node, Operators(node)))
  end loop

- Candidates are added to *nodes* representing the queue structure

Implementation of search

- A **search tree node** is a data-structure constituting part of a search tree



- Expand function – applies Operators to the state represented by the search tree node. Together with Queuing-fn it fills the attributes.

Uninformed search methods

- rely only on the information available in the problem definition
  - **Breadth first search**
  - **Depth first search**
  - **Iterative deepening**
  - **Bi-directional search**

For the minimum cost path problem:

- **Uniform cost search**

Search methods

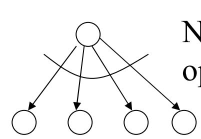
Properties of search methods :

- **Completeness.**
  - Does the method find the solution if it exists?
- **Optimality.**
  - Is the solution returned by the algorithm optimal? Does it give a minimum length path?
- **Space and time complexity.**
  - How much time it takes to find the solution?
  - How much memory is needed to do this?

Parameters to measure complexities.

- **Space and time complexity.**
  - **Complexities** are measured in terms of parameters:
    - b – maximum branching factor
    - d – depth of the optimal solution
    - m – maximum depth of the state space

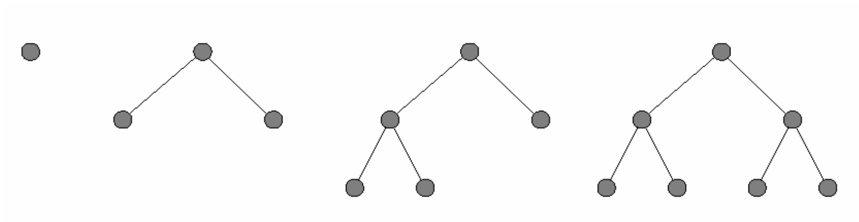
Branching factor

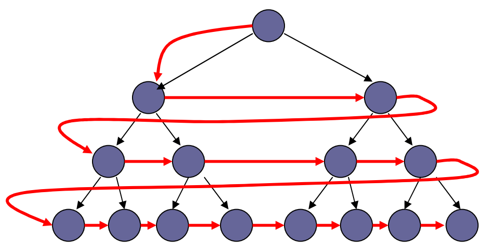


Number of operators

Breadth first search (BFS)

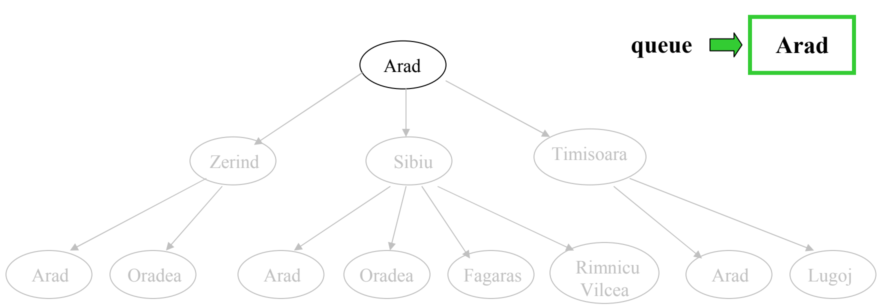
- **The shallowest node is expanded first**





Breadth-first search

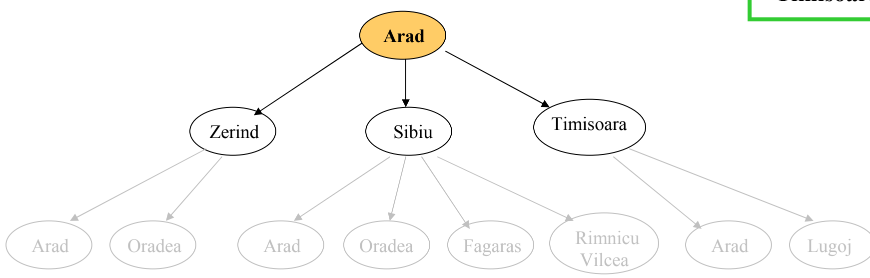
- **Expand the shallowest node first**
- Implementation: put successors to the end of the queue (FIFO)



Breadth-first search

queue

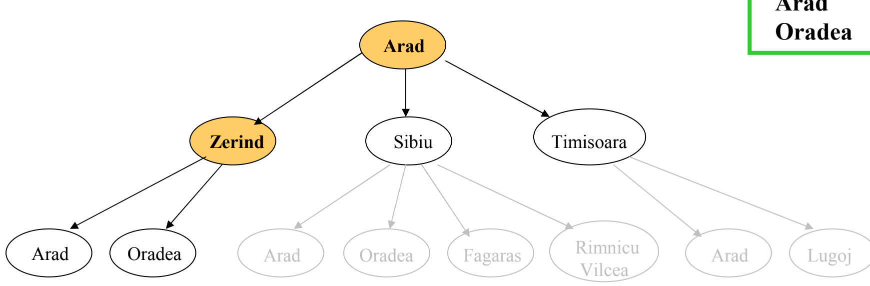
Zerind
Sibiu
Timisoara



Breadth-first search

queue

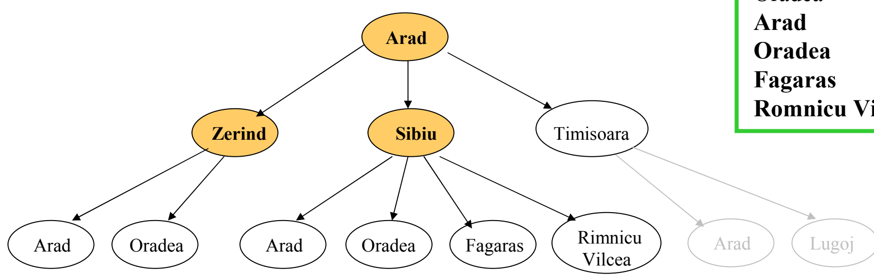
Sibiu
Timisoara
Arad
Oradea



Breadth-first search

queue

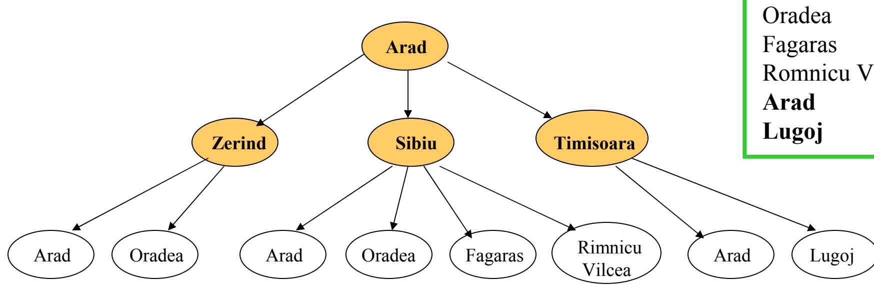
Timisoara
Arad
Oradea
Arad
Oradea
Fagaras
Romnicu Vilcea



Breadth-first search

queue

Arad
Oradea
Arad
Oradea
Fagaras
Romnicu Vilcea
Arad
Lugoj



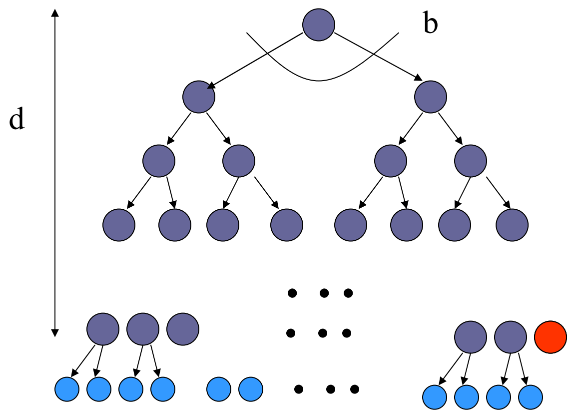
Properties of breadth-first search

- **Completeness: ?**
- **Optimality: ?**
- **Time complexity: ?**
- **Memory (space) complexity: ?**
  - **For complexities use:**
    - b – maximum branching factor
    - d – depth of the optimal solution
    - m – maximum depth of the search tree

Properties of breadth-first search

- **Completeness: Yes.** The solution is reached if it exists.
- **Optimality: Yes**, for the shortest path.
- **Time complexity: ?**
- **Memory (space) complexity: ?**

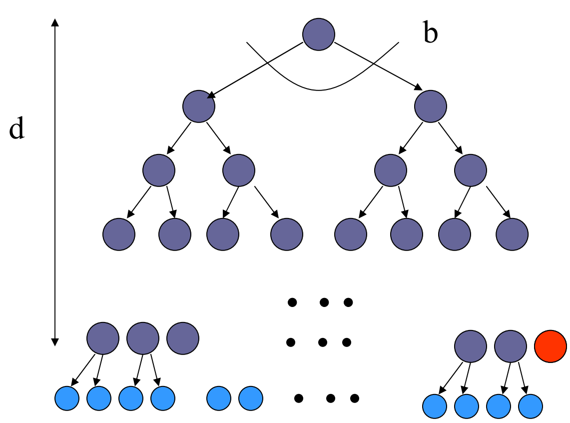
BFS – time complexity



depth	number of nodes	
0	1	
1	$2^1=2$	
2	$2^2=4$	
3	$2^3=8$	
d	2^d	(b^d)
d+1	2^{d+1}	(b^{d+1})

Total nodes: **?**

BFS – time complexity



depth	number of nodes	
0	1	
1	$2^1=2$	
2	$2^2=4$	
3	$2^3=8$	
d	2^d	(b^d)
d+1	2^{d+1}	(b^{d+1})

Expanded nodes: $O(b^d)$

Total nodes: $O(b^{d+1})$

Properties of breadth-first search

- **Completeness:** **Yes.** The solution is reached if it exists.
- **Optimality:** **Yes**, for the shortest path.
- **Time complexity:**

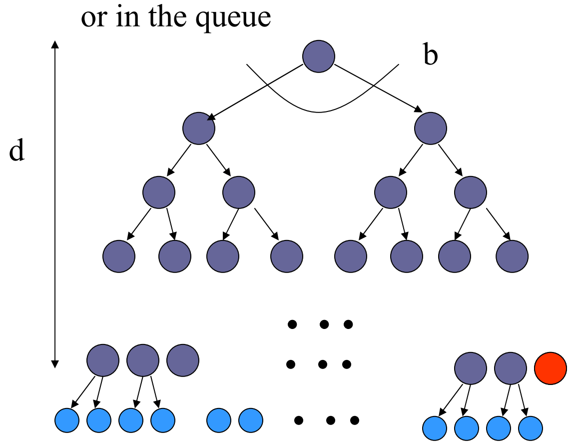
$$1 + b + b^2 + \ldots + b^d = O(b^d)$$

exponential in the depth of the solution *d*

- **Memory (space) complexity: ?**

BFS – memory complexity

- Count nodes kept in the tree structure or in the queue

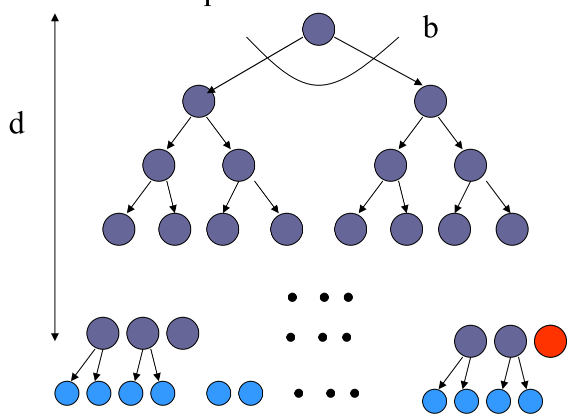


depth	number of nodes
0	1
1	$2^1=2$
2	$2^2=4$
3	$2^3=8$
d	2^d (b^d)
d+1	2^{d+1} (b^{d+1})

Total nodes: **?**

BFS – memory complexity

- Count nodes kept in the tree structure or in the queue



depth	number of nodes
0	1
1	$2^1=2$
2	$2^2=4$
3	$2^3=8$
d	2^d (b^d)
d+1	2^{d+1} (b^{d+1})

Expanded nodes: $O(b^d)$

Total nodes: $O(b^{d+1})$

Properties of breadth-first search

- **Completeness:** **Yes.** The solution is reached if it exists.
- **Optimality:** **Yes**, for the shortest path.
- **Time complexity:**

$$1 + b + b^2 + \ldots + b^d = O(b^d)$$

exponential in the depth of the solution *d*

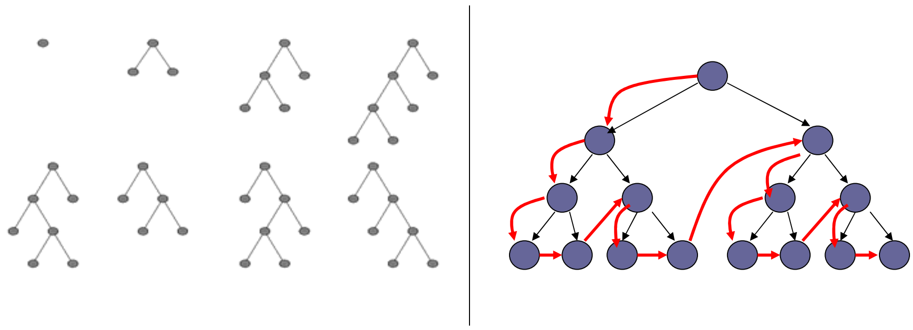
- **Memory (space) complexity:**

$$O(b^d)$$

nodes are kept in the memory

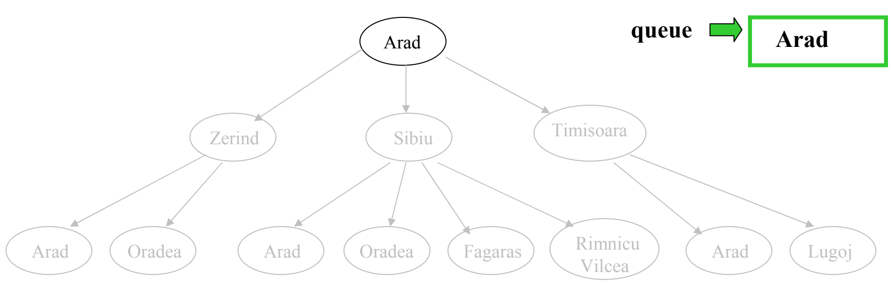
Depth-first search (DFS)

- **The deepest node is expanded first**
- Backtrack when the path cannot be further expanded

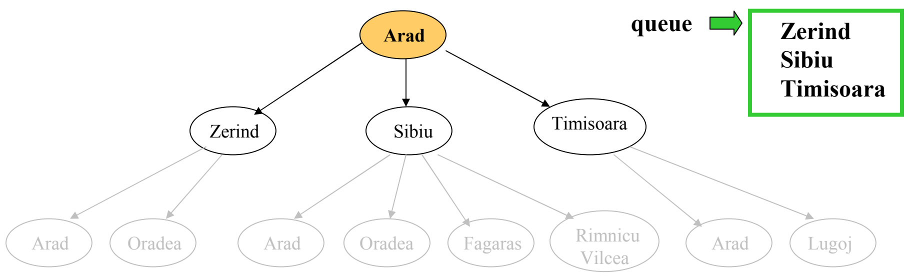


Depth-first search

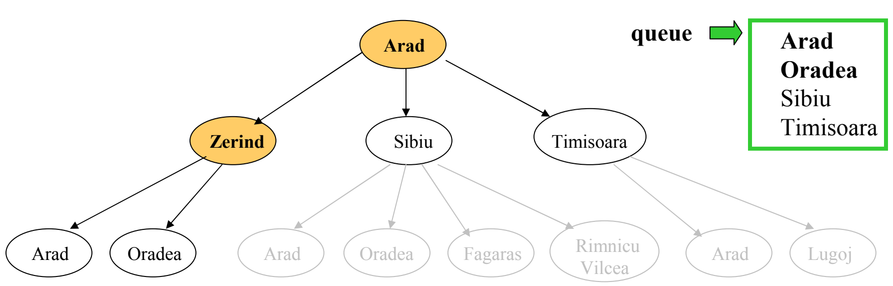
- **The deepest node is expanded first**
- Implementation: put successors to the beginning of the queue



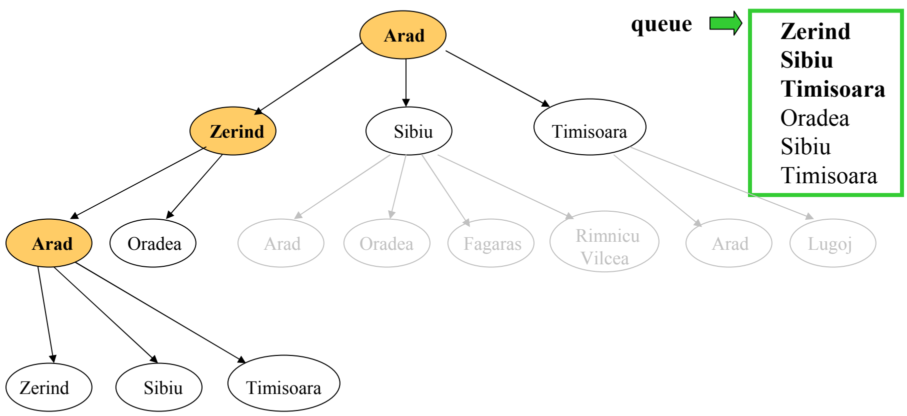
Depth-first search



Depth-first search



Depth-first search



Note: Arad – Zerind – Arad cycle

Properties of depth-first search

- **Completeness: Does it always find the solution if it exists?**
- **Optimality: ?**
- **Time complexity: ?**
- **Memory (space) complexity: ?**

Properties of depth-first search

- **Completeness:** **No.** Infinite loops can occur.
  Infinite loops imply -> Infinite depth search tree.
- **Optimality: does it find the minimum length path ?**
- **Time complexity: ?**
- **Memory (space) complexity: ?**

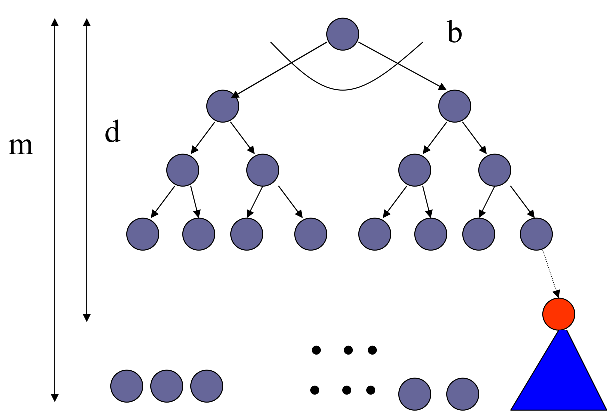
Properties of depth-first search

- **Completeness:** **No.** Infinite loops can occur.
- **Optimality:** **No.** Solution found first may not be the shortest possible.
- **Time complexity: ?**
- **Memory (space) complexity: ?**

Properties of depth-first search

- **Completeness:** No. Infinite loops can occur.
- **Optimality:** No. Solution found first may not be the shortest possible.
- **Time complexity: ?**
- **Memory (space) complexity: ?**

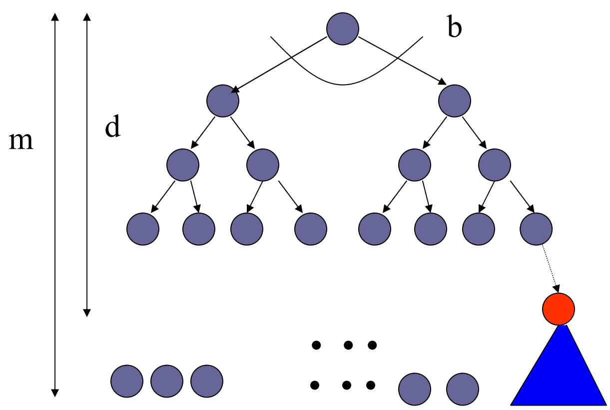
DFS – time complexity



depth	number of nodes
0	1
1	$2^1=2$
2	$2^2=4$
3	$2^3=8$
•	• •
d	2^d
•	• •
m	2^m-2^{m-d}

Complexity:

DFS – time complexity



depth	number of nodes
0	1
1	$2^1=2$
2	$2^2=4$
3	$2^3=8$
•	• •
d	2^d
•	• •
m	2^m-2^{m-d}

Complexity: $O(b^m)$

Properties of depth-first search

- **Completeness:** **No.** Infinite loops can occur.
- **Optimality:** **No.** Solution found first may not be the shortest possible.
- **Time complexity:**

  $O(b^m)$

  exponential in the maximum depth of the search tree *m*
- **Memory (space) complexity: ?**

Properties of depth-first search

- **Completeness: No.** Infinite loops can occur.
- **Optimality: No.** Solution found first may not be the shortest possible.
- **Time complexity:**

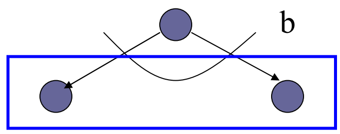
  $O(b^m)$

  exponential in the maximum depth of the search tree *m*
- **Memory (space) complexity: ?**

DFS – memory complexity

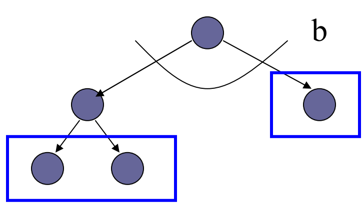
depth	number of nodes kept
0	1

DFS – memory complexity



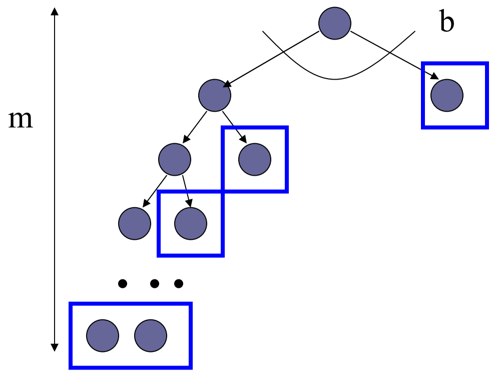
depth	number of nodes kept
0	0
1	2 = b

DFS – memory complexity



depth	number of nodes kept
0	0
1	1 = (b-1)
2	2 = b

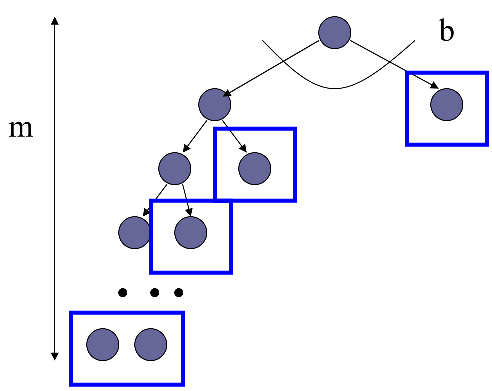
DFS – memory complexity



depth	number of nodes kept
0	0
1	1
2	1
3	1
• • •	
m	2=b

Complexity:

DFS – memory complexity



depth	number of nodes kept
0	0
1	1=(b-1)
2	1= (b-1)
3	1 =(b-1)
• • •	
m	2=b

Complexity: $O(bm)$

Properties of depth-first search

- **Completeness:** **No.** Infinite loops can occur.
- **Optimality:** **No.** Solution found first may not be the shortest possible.
- **Time complexity:**

  $O(b^m)$

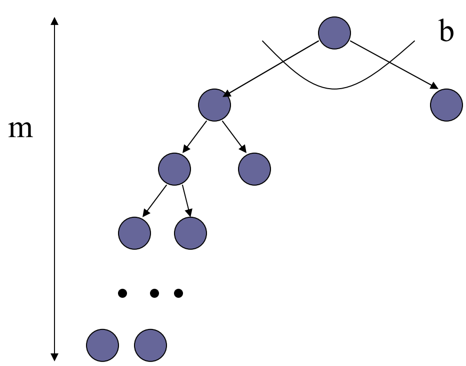
  exponential in the maximum depth of the search tree *m*
- **Memory (space) complexity:**

  $O(bm)$

  linear in the maximum depth of the search tree *m*

DFS – memory complexity

Count nodes kept in the tree structure or the queue



depth	number of nodes
0	1
1	2 = b
2	2
3	2
• • •	
m	2

Total nodes: $O(bm)$

Properties of depth-first search

- **Completeness:** **No.** Infinite loops can occur.
- **Optimality:** **No.** Solution found first may not be the shortest possible.
- **Time complexity:**

  $O(b^m)$

  exponential in the maximum depth of the search tree *m*

- **Memory (space) complexity:**

  $O(bm)$

the tree size we need to keep is linear in the maximum depth of the search tree *m*