CS 1571 Introduction to AI
Lecture 3

Problem solving by searching

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

A search problem

Many interesting problems in science and engineering are solved using search

A search problem is defined by:

- **A search space:**
  - The set of objects among which we search for the solution
    Examples: routes between cities, or n-queens configuration
- **A goal condition**
  - Characteristics of the object we want to find in the search space?
  - **Examples:**
    - Path between cities A and B
    - Non-attacking n-queen configuration

Graph Search Problems

Search problems can be often represented as graph search problems:

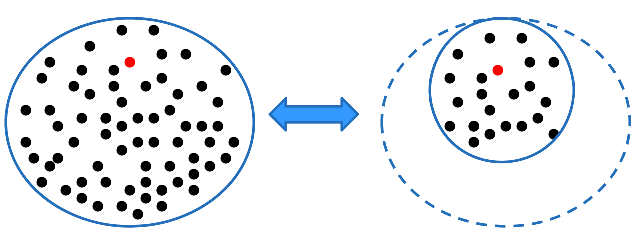
- **Initial state**
  - State (configuration) we start to search from (e.g. start city, initial game position)
- **Operators**:
  - Transform one state to another (e.g. valid connections between cities, valid moves in Puzzle 8)
- **Goal condition:**
  - Defines the target state (destination, winning position)

Search space is now defined indirectly through**:**

The initial state + Operators

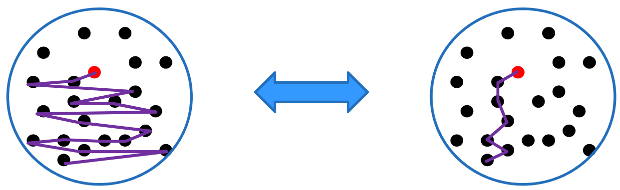
Search

- **Search:** The process of exploration of the search space
- **Design goal:** We want the search to be as efficient as possible
- **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



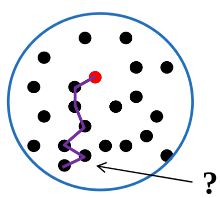
Search

- **Search:** The process of exploration of the search space
- **Design goal:** We want the search to be as efficient as possible
- **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



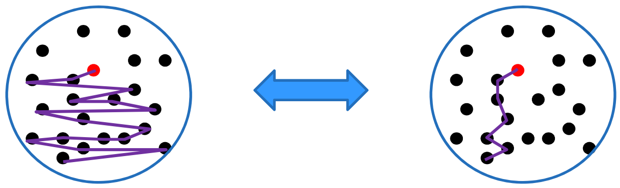
Search

- **Search:** The process of exploration of the search space
- **Design Goal:** We want the search to be as efficient as possible
- **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**



This lecture

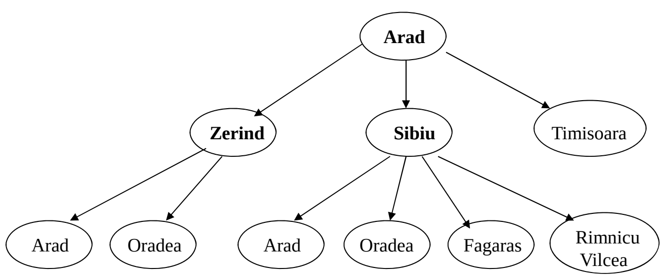
- **Focus on:**
  - **Methods used to explore (traverse) the search space**



Search process

Exploration of the state space through successive application of operators from the initial state

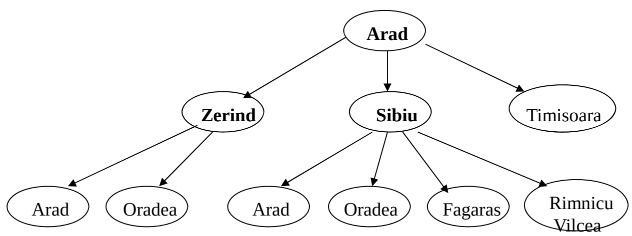
- **Search tree = structure representing the exploration trace**
- Built on-line during the search process
- Branches correspond to explored paths, and leaf nodes to the exploration fringe



Search tree

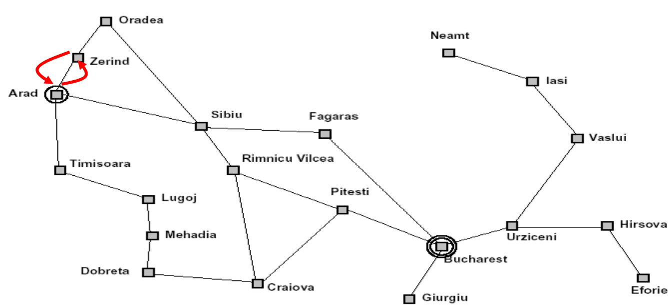
- A **search tree = (search) exploration trace**
  - **different from the graph representation of the problem**
  - states can repeat in the search tree

Graph

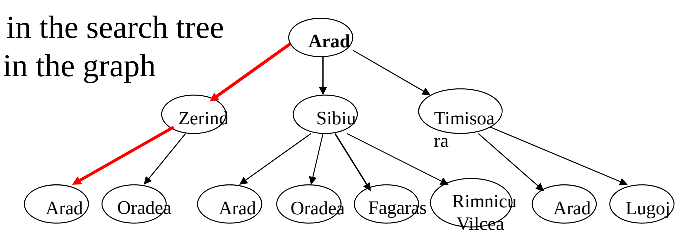


Search tree

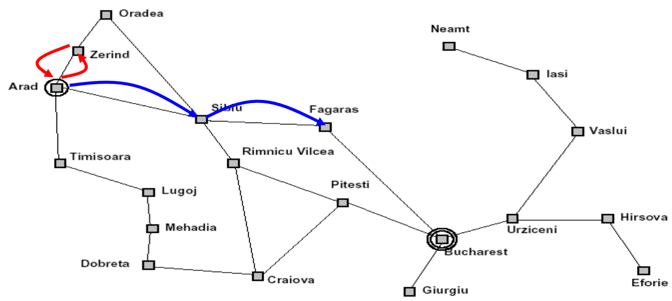
Search tree



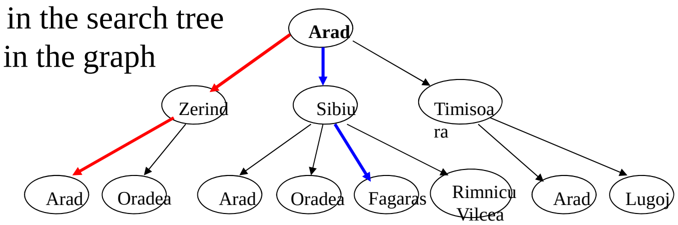
A branch in the search tree
= path in the graph



Search tree



A branch in the search tree
= path in the graph

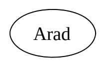


General search algorithm

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

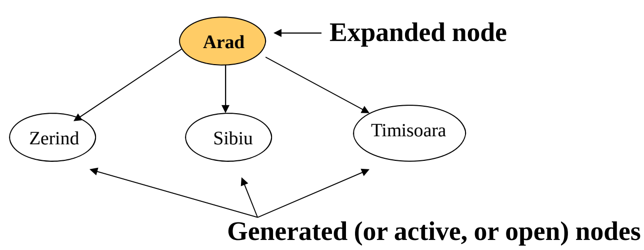
General search algorithm

General-search (*problem, strategy*)
initialize the search tree with the initial state of *problem*
loop
  if there are no candidate states to explore next **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



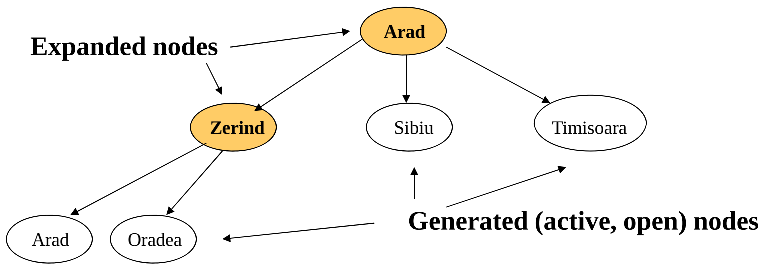
General search algorithm

General-search (*problem, strategy*)
initialize the search tree with the initial state of *problem*
loop
  if there are no candidate states to explore next **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



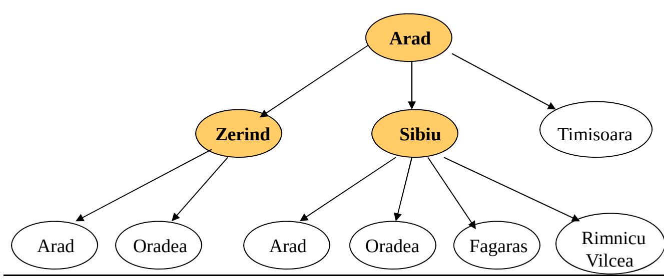
General search algorithm

General-search (*problem, strategy*)
initialize the search tree with the initial state of *problem*
loop
  if there are no candidate states to explore next **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



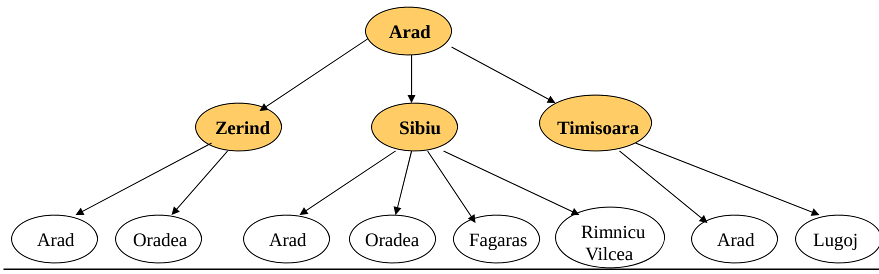
General search algorithm

General-search (*problem, strategy*)
initialize the search tree with the initial state of *problem*
loop
  if there are no candidate states to explore next **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 next **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 next **return** failure
  choose a leaf node of the tree to expand next **according to a *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

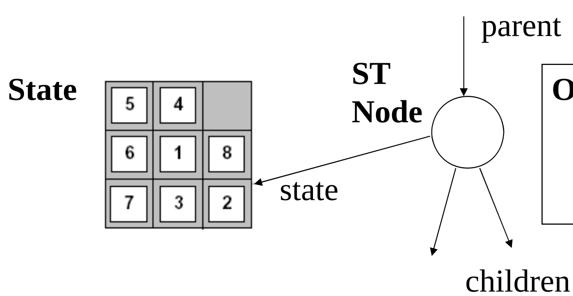
- Search methods can be implemented using the **queue** structure

General search (*problem*, Queuing-fn)
  nodes ← Make-queue(Make-node(Initial-state(*problem*)))
  loop
    if nodes is empty **then return** failure
    node ← Remove-node(*nodes*)
    if Goal-test(*problem*) applied to State(*node*) is satisfied **then return** node
    nodes ← 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 that is a part of the search tree



Other attributes:
- state value (cost)
- depth
- path cost

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

Uninformed search methods

- Search techniques that 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

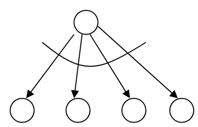
Properties of search methods :

- **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.**
  - **Complexity** is measured in terms of the following tree parameters:
    - *b* – maximum branching factor
    - *d* – depth of the optimal solution
    - *m* – maximum depth of the state space

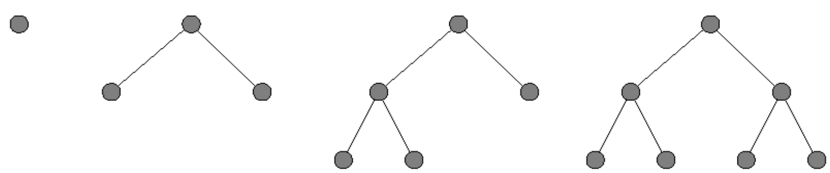
Branching factor

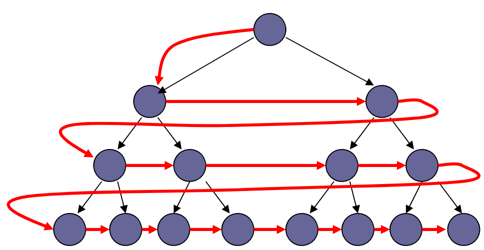


The number of applicable operators

Breadth first search (BFS)

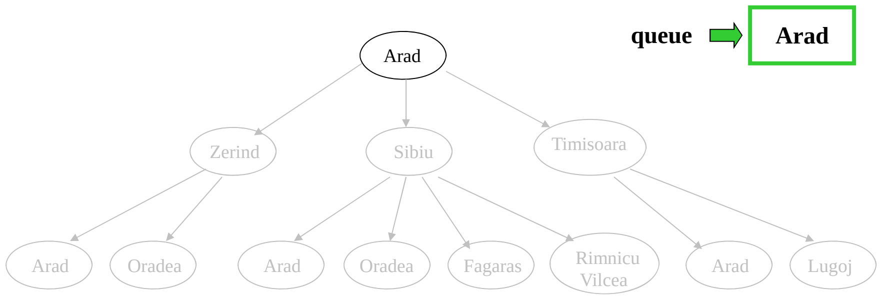
- **The shallowest node is expanded first**



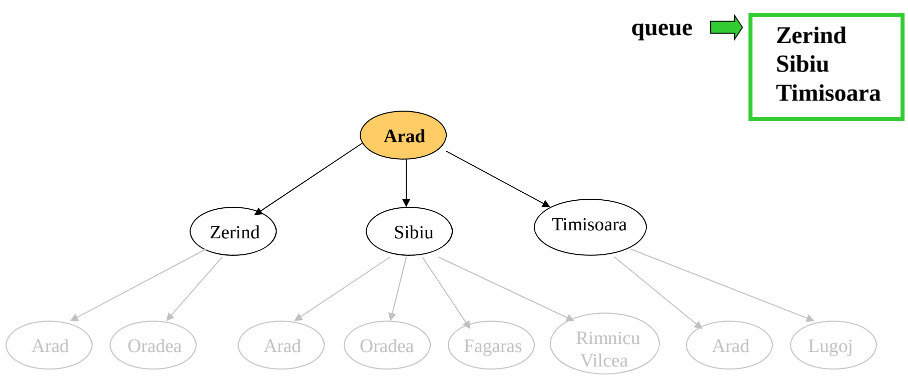


Breadth-first search

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



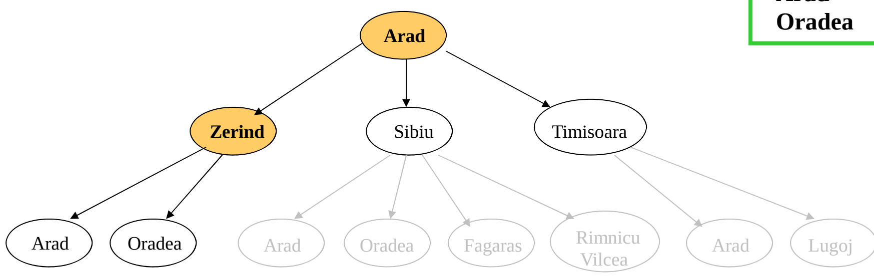
Breadth-first search



Breadth-first search

queue ➡

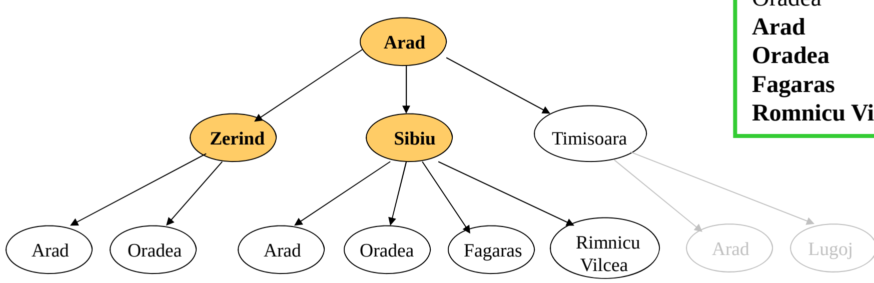
Sibiu
Timisoara
Arad
Oradea



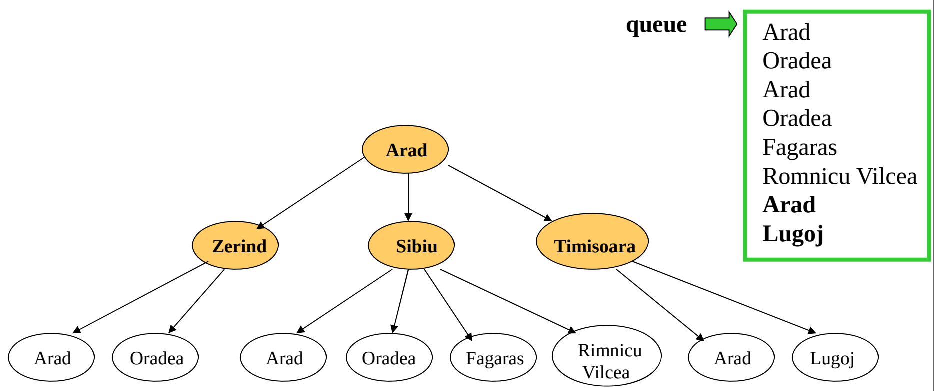
Breadth-first search

queue ➡

Timisoara
Arad
Oradea
Arad
Oradea
Fagaras
Romnicu Vilcea



Breadth-first search



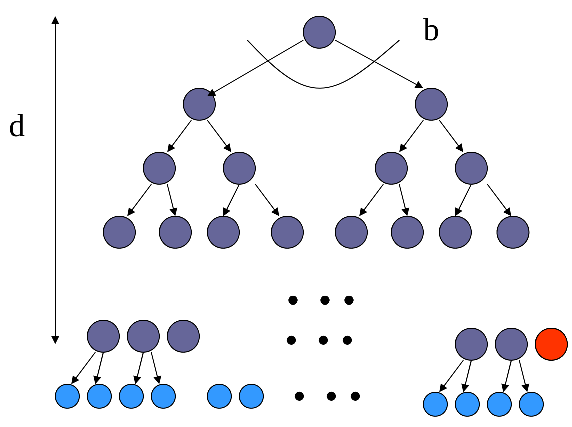
Properties of breadth-first search

- **Completeness: ?**
- **Optimality: ?**
- **Time complexity: ?**
- **Memory (space) complexity: ?**
  - **For complexity 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})

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:**

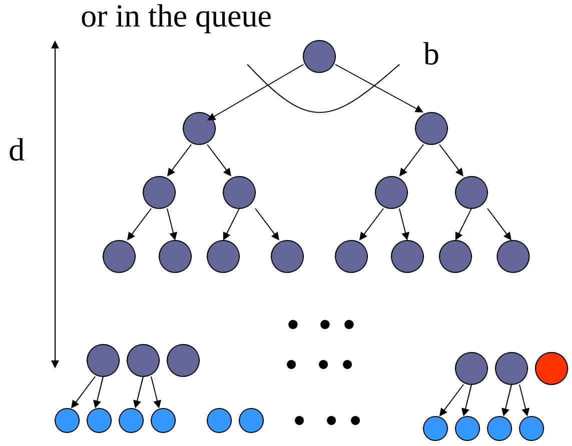
$$1 + b + b^2 + \ldots + b^d = O(b^d)$$

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})

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*

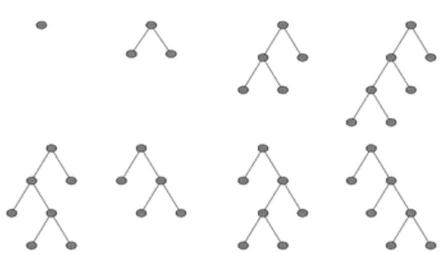
- **Memory (space) complexity:**

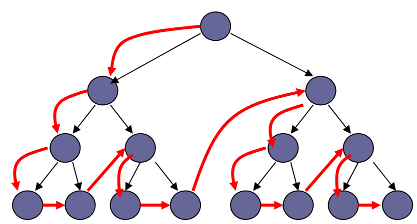
$$O(b^d)$$

nodes are kept in the memory

Depth-first search (DFS)

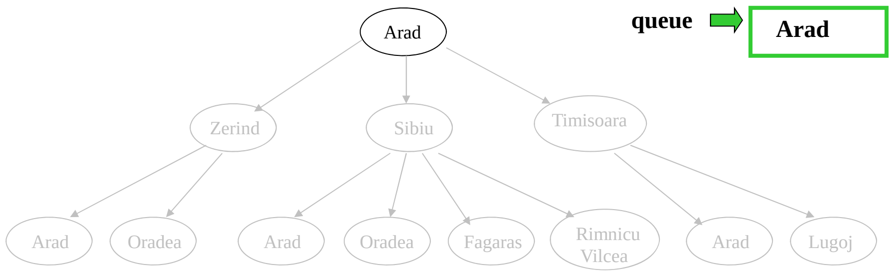
- **The deepest node is expanded first**
- Backtrack when the path cannot be further expanded



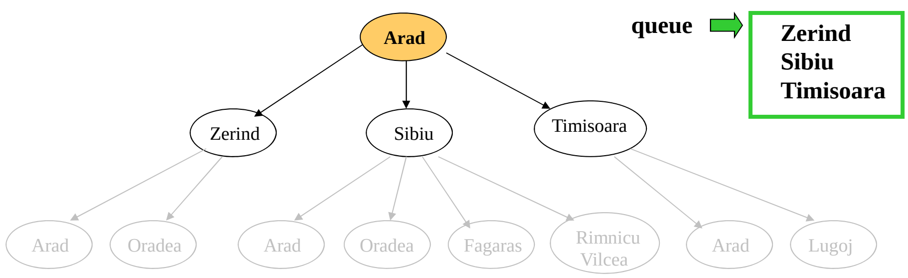


Depth-first search

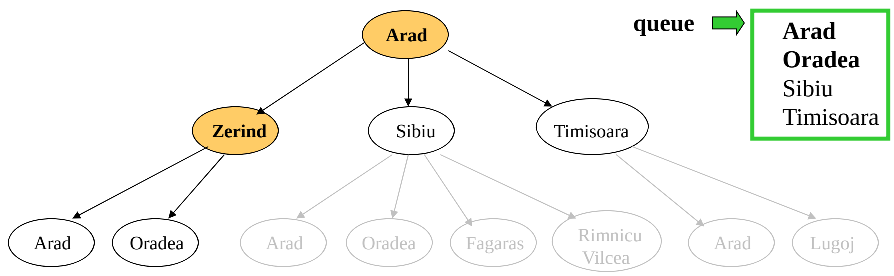
- **The deepest node is expanded first**
- Implementation: put successors to the beginning of the queue



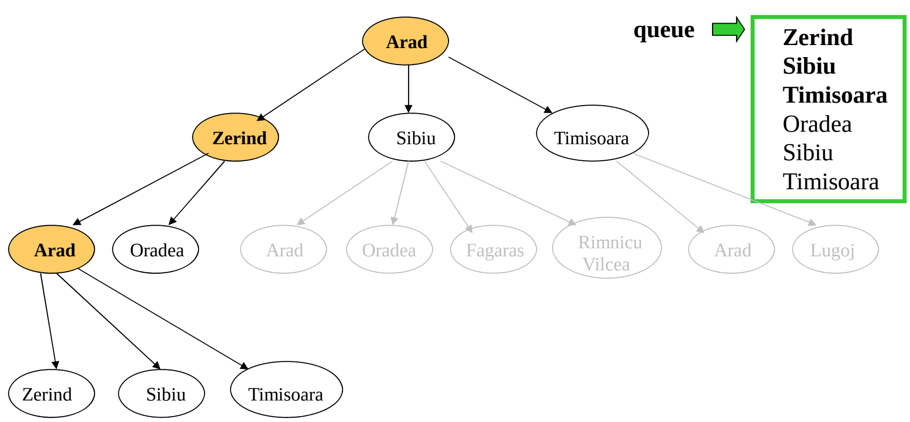
Depth-first search



Depth-first search



Depth-first search

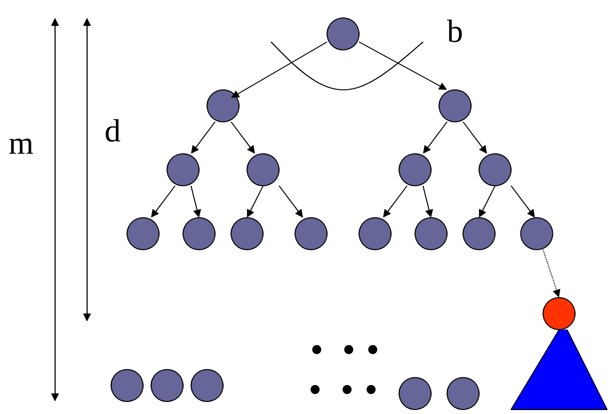


Note: Arad – Zerind – Arad cycle

Properties of depth-first search

- **Completeness:** **No.** when no limit Infinite loops can occur.
  May be when the max depth limit is set
  - depends on how it is set
- **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: $O(b^m)$

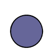
Properties of depth-first search

- **Completeness:** **No.** when no limit Infinite loops can occur.
  May be when the max depth limit is set
  - depends on how it is set
- **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: ?**

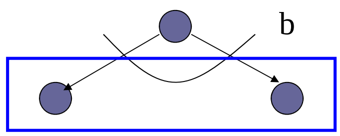
DFS – memory complexity



b

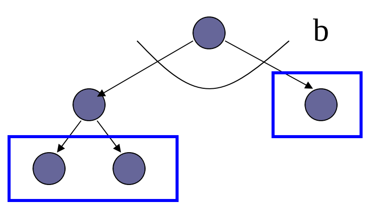
depth	number of nodes kept
0	1

DFS – memory complexity



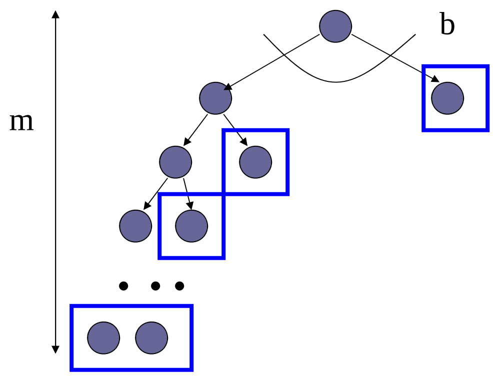
depth	number of nodes kept
0	0
1	2 = b

DFS – memory complexity



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

DFS – memory complexity

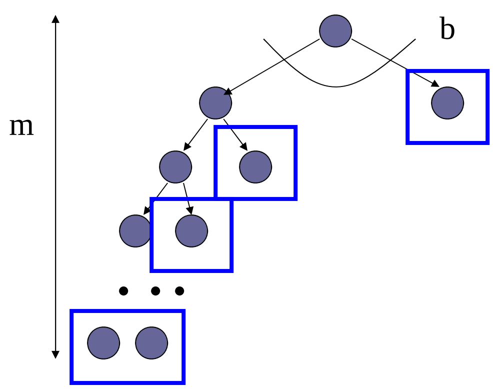


depth	number of nodes kept
0	0
1	1
2	1
3	1
• • •	
m	2=b

Complexity:

CS 1571 Intro to AI M. Hauskrecht

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: $O(bm)$

CS 1571 Intro to AI M. Hauskrecht

Properties of depth-first search

- **Completeness:** **No.** when no limit Infinite loops can occur.
  May be when the max depth limit is set
  - depends on how it is set
- **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*