CSC340/AI320 - Artificial Intelligence
Fall 2022

Lab 4
Blind Searching Algorithms

INSTRUCTOR: DR. DIAA SALAMA ABDEL MONEIM
ENG. NOHA EL MASRY
ENG. NOUR ELHUDA ASHRAF

Problem-solving:

Solving problems by searching, in which we see how an **agent** can find a **sequence of actions** that achieves its goals **when no single action will do**.

We will describe one kind of **goal-based agent** called a **problem-solving agent**.

Problem-solving agents use atomic representations, that is, states of the world are considered as wholes, with no internal structure visible to the problem-solving algorithms.

We will see several **uninformed (Blind) search algorithms**, algorithms that are given **no information about the problem** other than its definition.

We limit ourselves to the **simplest kind of task environment**, for which the **solution** to a problem is always a **fixed sequence of actions**.

Goal formulation, based on the current situation and the agent's performance measure, is the first step in problem solving.

Problem formulation is the process of deciding what actions and states to consider, given a goal.

Search is the process of looking for a sequence of actions that reaches the goal.

The environment of the problem must be:

⇒ **Observable** (agent always knows the current state).

⇒ **Discrete** (at any given state there are only finitely many actions to choose from).

⇒ **Known** (the agent knows which states are reached by each action).

⇒ **Deterministic** (each action has exactly one outcome).

A problem can be defined formally by five components:

⇒ **Initial state**

⇒ **Description of the possible actions available to the agent**

⇒ **Description of what each action does** (successor function / transition model)

⇒ **The goal test**

⇒ **Path cost**

A **solution** is a sequence of actions leading from the initial state to a goal state.

An **optimal solution** is a solution with minimal path cost.

State-Space vs Search Tree

- The **state-space** is a **graph**, while the **search tree** is a **tree** (each node has a single parent).
- **Links in the state-space** might be **bidirectional**, while **links in a tree** are **unidirectional** (flow from parent to child only).
- Each **node in the state space** represents a **state**, while **each node in a tree** represents a **state** + **path from root** (initial state).
- A **state in a state space can appear only once**, while **it can appear in a tree multiple times** in case it is reachable from multiple paths.
- **A tree is a data structure to search the state-space**.

The possible action sequences starting at the initial state form a search tree with the initial state at the root; the branches are actions and the nodes correspond to states in the state space of the problem.

Steps for searching a state space (generating a search tree):

1. Test whether this is a goal state.
2. Expand the current state.
3. Generate a new set of states.
4. Back to step 1.

N.B The process of expanding the current state continues until either a solution is found or there are no more states to expand.

➢ Measuring problem-solving performance

We can evaluate an algorithm's performance in four ways:

1. **Completeness**: Is the algorithm guaranteed to find a solution when there is one?
2. **Optimality**: Does the strategy find the optimal solution?
3. **Time complexity**: How long does it take to find a solution?
4. **Space complexity**: How much memory is needed to perform the search?

Uninformed search (also called **blind search**) means that the strategies have **no additional information** about states beyond that provided in the problem definition.

All search strategies are distinguished by the order in which nodes are expanded.

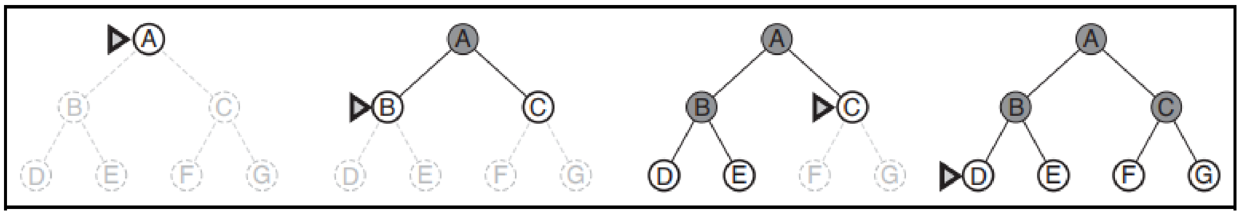
Breadth First Search (BFS)

Expand the shallowest unexpanded node first.

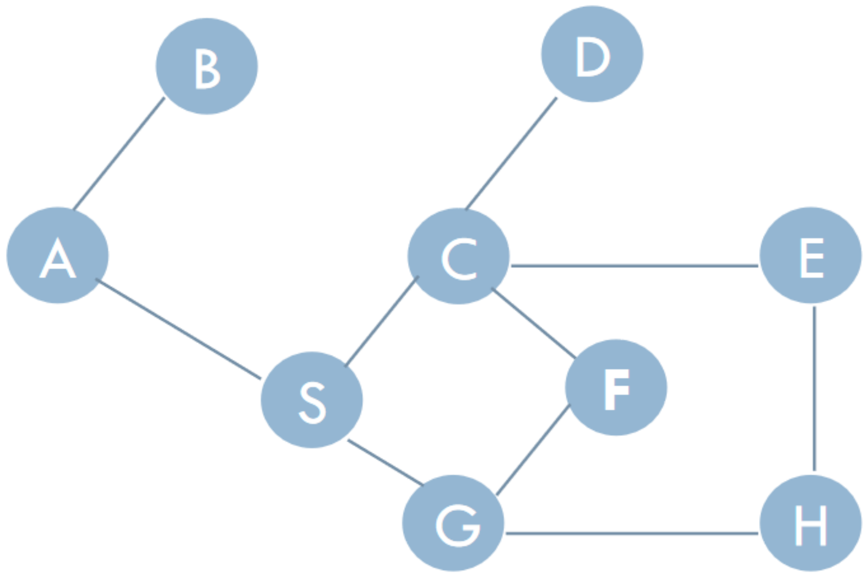
In general, all the nodes are expanded at a given depth in the search tree before any nodes at the next level are expanded.

This is achieved very simply by using a **FIFO** queue for the frontier. Thus, new nodes (which are always deeper than their parents) go to the back of the queue, and old nodes, which are shallower than the new nodes, get expanded first.

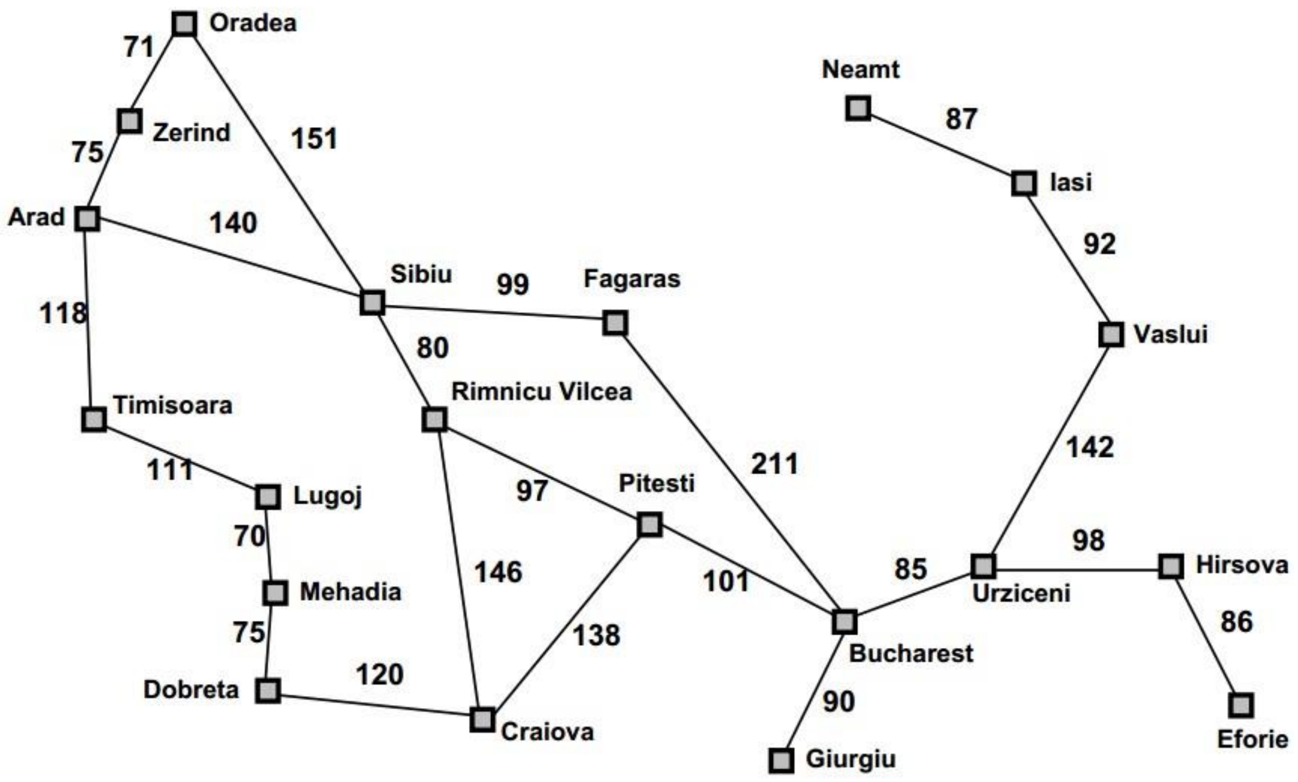
Example 1:



Example 2:



Example 3 (with code):



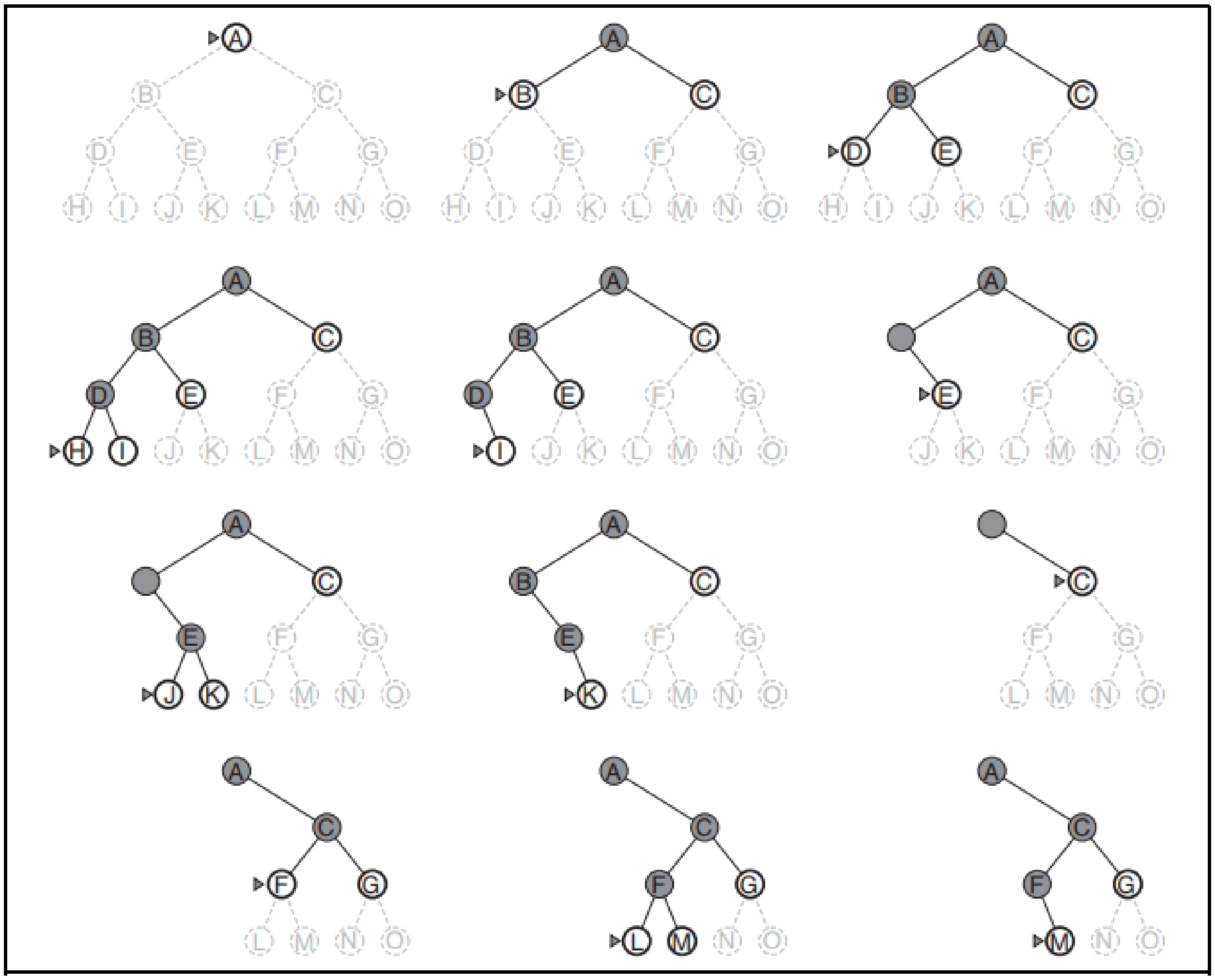
Depth First Search (DFS)

Always expands the deepest node in the current frontier of the search tree.

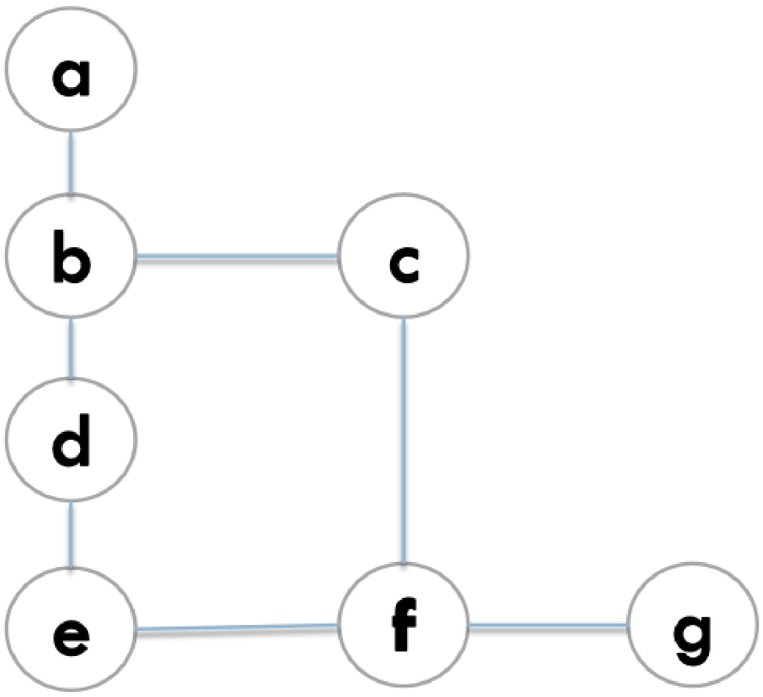
Explores the first (leftmost) possible option first and then recursively explore its first (leftmost) children. Keep doing this until the goal is reached or until forced to backtrack.

This is achieved very simply by using a **LIFO** queue, a LIFO queue means that the most recently generated node is chosen for expansion. This must be the deepest unexpanded node because it is one deeper than its parent, which, in turn, was the deepest unexpanded node when it was selected.

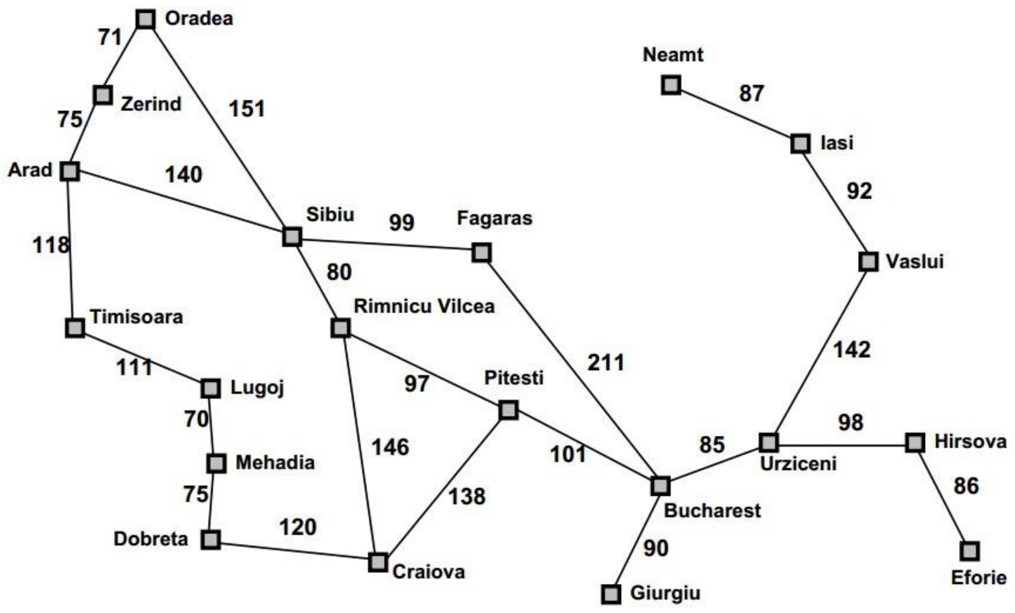
Example 1:



Example 2:



Example 3 (with code):



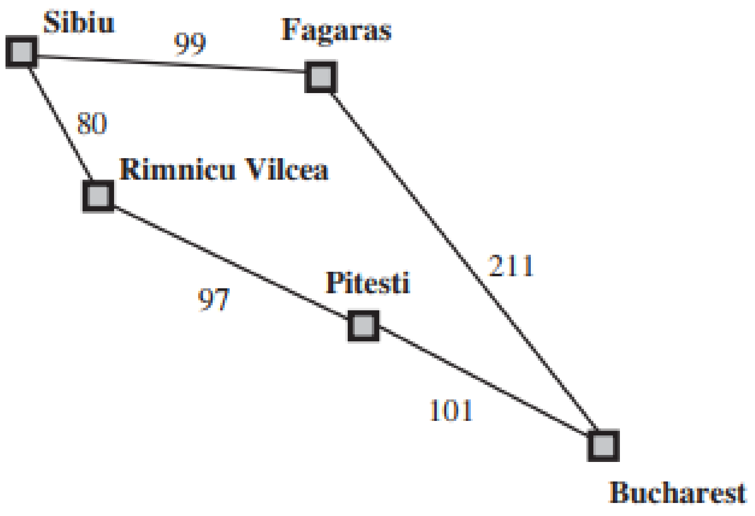
Uniform Cost Search (UCS)

When all step costs are equal, **breadth-first search** is optimal because it always expands the shallowest unexpanded node.

Uniform-cost search expands the node n with the lowest path cost g(n).

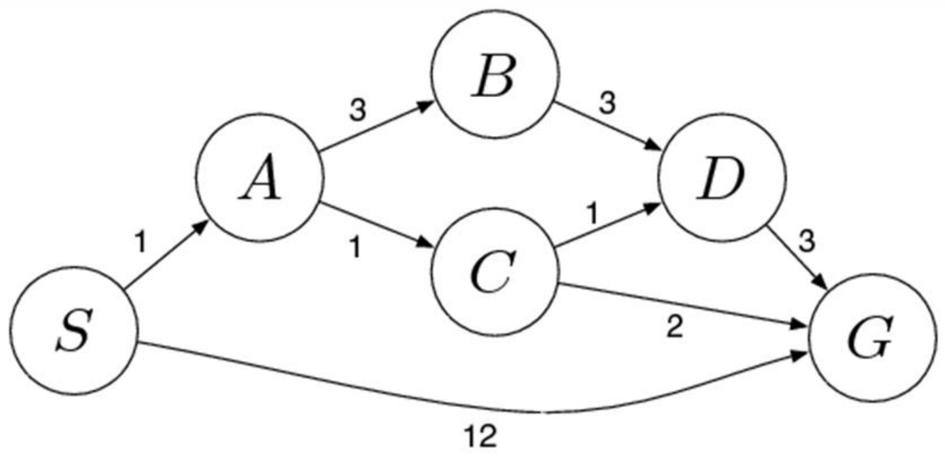
This is done by storing the frontier as a **priority queue** ordered by g.

Example 1:

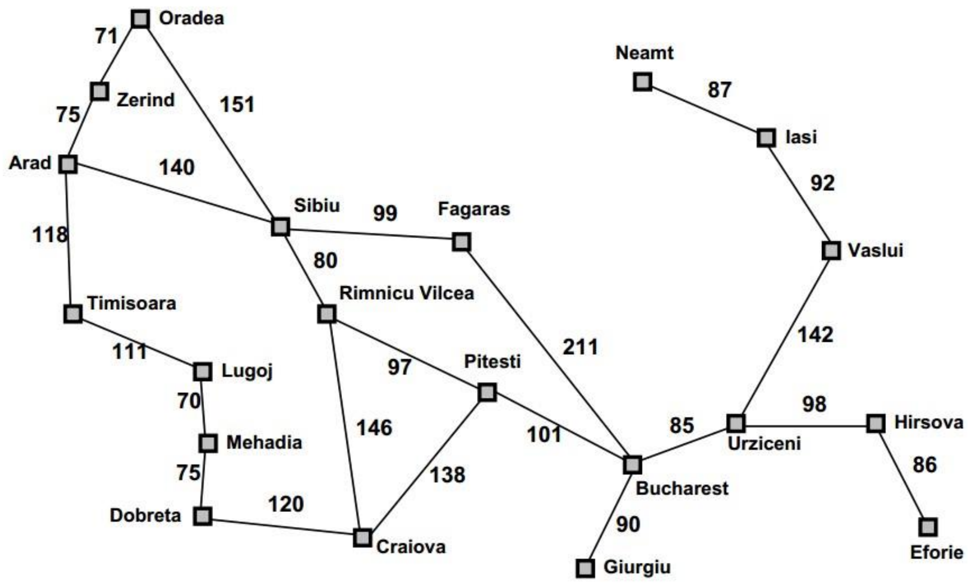


Example 2:

for the following graph, starting from state 'A', reach the goal state using UCS. Draw down the associated searching trees.



Example 3 (with code):



Uninformed Searching Algorithms Comparison

The below figure depicts a comparison of search strategies in terms of the four evaluation criteria set. This comparison is for tree-search versions. For graph searches, the main differences are that depth-first search is complete for finite state spaces and that the space and time complexities are bounded by the size of the state space

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Complete?	Yes^a	Yes^{a,b}	No	No	Yes^a	Yes^{a,d}
Time	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(b^m)$	$O(b^\ell)$	$O(b^d)$	$O(b^{d/2})$
Space	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(bm)$	$O(b\ell)$	$O(bd)$	$O(b^{d/2})$
Optimal?	Yes^c	Yes	No	No	Yes^c	Yes^{c,d}

b is the branching factor; d is the depth of the shallowest solution; m is the maximum depth of the search tree; l is the depth limit. Superscript caveats are as follows: a complete if b is finite; b complete if step costs ≥ ϵ for positive ϵ; c optimal if step costs are all identical; d if both directions use breadth-first search.