THUẬT TOÁN ỨNG DỤNG
SOLVING PROBLEM BY SEARCHING
Lecture 1: Giới Thiệu

T.S Hà Minh Hoàng
Th.S Nguyễn Minh Anh

Phenikaa University

Last Update: Ngày 17 tháng 1 năm 2023

Outline

Course Elements

- 10 buổi lí thuyết + 10 buổi thực hành
- Assessment: including multiple homeworks, 1 midterm (GK) và 1 project (BTL)
- Ratio: 10% CC, 40% GK, and 50% BTL

Course Material

- **Slides**: can be found on Canvas
- **Textbook**:
  Russell, S. J. (2021). **Artificial intelligence a modern approach**, Fourth Edition, Pearson Education, Inc..
- Consider **every web resource** highly suspect until you have reason to believe better of it

Course Outline

Part I: Graph and Search

- Graph theory and basic search algorithms (1)
- Uninformed search strategies (1)
- Informed (Heuristic) search strategies (1)

Part II: Searching in Complex Environments

- Constraint Satisfaction Problems (CSP) (1)
- Recursive / Backtracking search + heuristics for CSPs (2)
- Local search for CSPs

Part III: Searching in Optimization Problems

- Greedy algorithm, heuristics, metaheuristics(1)
- Problems: Knapsack, TSP, VRP (1)

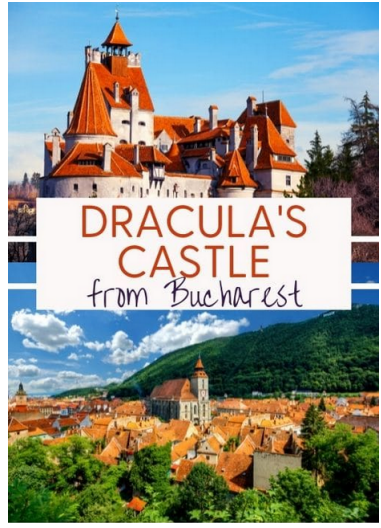
Further Remark

- We use ISO standard C++ 14 or newer in developing algorithms
- It is important to sketch first the program on a piece of paper. This is a useful step in solving a programming task. The design is a human brain activity not a computer activity and the best way not to skip this stage is to stay away from computer

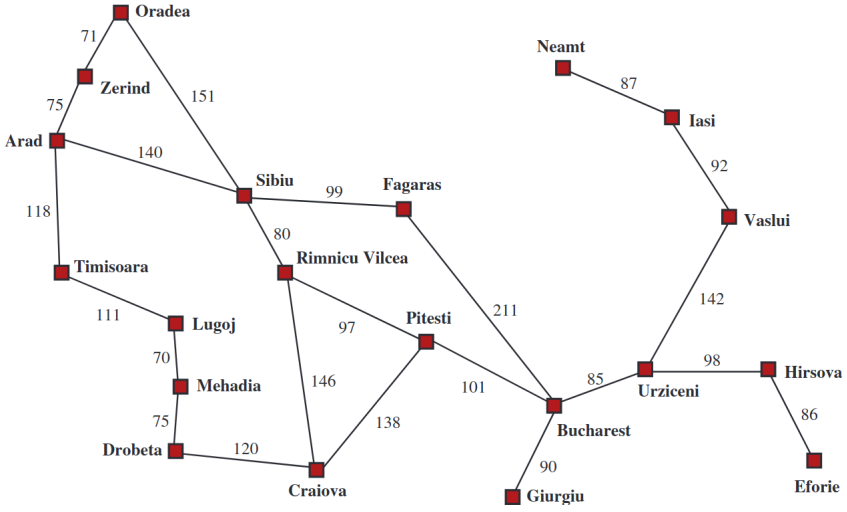
Outline

Romania vacation

Imagine an agent enjoying a touring vacation in Romania. The agent wants to take in the sights, improve its Romanian, enjoy the nightlife, avoid hangovers, and so on. Now, suppose the agent is currently in the city of Arad and has a nonrefundable ticket to fly out of Bucharest the following day.

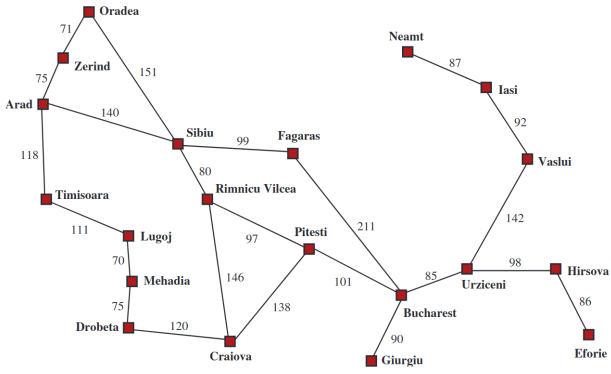


Romania vacation



Romania vacation

- Performance: Get from Arad to Bucharest as quickly as possible
- Environment: The map, with cities, roads, and guaranteed travel times
- Actions: Travel a road between adjacent cities.



Problem-solving process

Assume our agents always have access to information about the world, such as the map in the previous. With that information, the agent can follow this four-phase problem-solving process:

- **Goal formulation:** the agent adopts the goal of reaching Bucharest. Goals organize behavior by limiting the objectives and hence the actions to be considered.
- **Problem formulation:** the agent devises a description of the states and actions necessary to reach the goal—an abstract model of the relevant part of the world. For our agent, one good model is to consider the actions of traveling from one city to an adja- cent city, and therefore the only fact about the state of the world that will change due to an action is the current city.

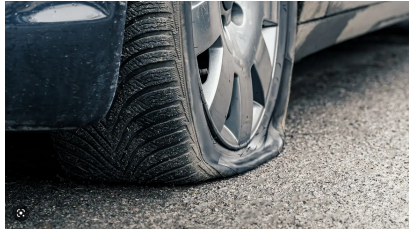
Problem-solving process

Assume our agents always have access to information about the world, such as the map in the previous. With that information, the agent can follow this four-phase problem-solving process:

- **Search:** before taking any action in the real world, the agent simulates sequences of actions in its model, searching until it finds a sequence of actions that reaches the goal. Such a sequence is called a solution. The agent might have to simulate multiple sequences that do not reach the goal, but eventually it will find a solution (such as going from Arad to Sibiu to Fagaras to Bucharest), or it will find that no solution is possible.
- **Execution:** the agent can now execute the actions in the solution, one at a time.

Problem types

- **Deterministic, fully observable** → single-state problem
  Agent knows exactly which state it will be in; solution is a sequence
- **Non-observable** → sensorless problem (conformant problem)
  Agent may have no idea where it is; solution is a sequence
- **Nondeterministic and/or partially observable** → contingency
  - percepts provide new information about current state
  - often interleave → search, execution
- **Unknown state space** → exploration problem problem



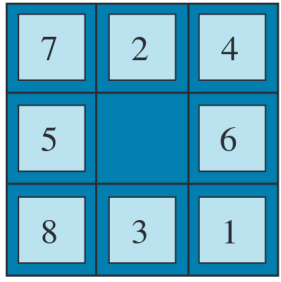
Search problem definition

A **problem** is defined by four items:

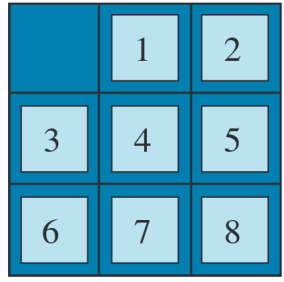
- **initial state:** e.g., Arad
- **actions** $S(x)$ = set of action-state pairs
  e.g., $S(Arad) = \{< Arad \rightarrow Sibiu >, < Arad \rightarrow Zerind >, ...\}$
- **goal check:**can be:
  - explicit, e.g., x = Bucharest
  - implicit, e.g., Checkmate(x)
- **path cost** (additive):
  - e.g., sum of distances, number of actions executed, etc.
  - c(x,a,y) is the step cost, assumed to be ≥ 0

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

Example: The 8-puzzle



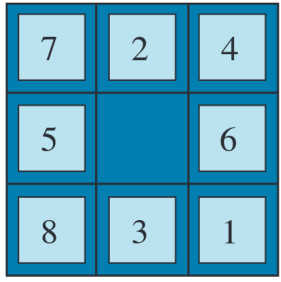
Start State



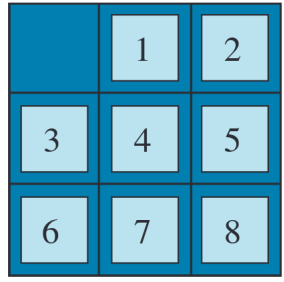
Goal State

- **initial state: ?**
- **actions: ?**
- **goal check: ?**
- **path cost: ?**

Example: The 8-puzzle



Start State

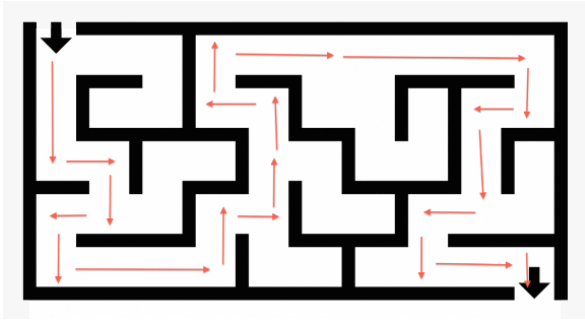


Goal State

- **initial state:** locations of tiles
- **actions:** move blank left, right, up, down
- **goal check:** = goal state (given)
- **path cost:** 1 per move

Other real-life problems

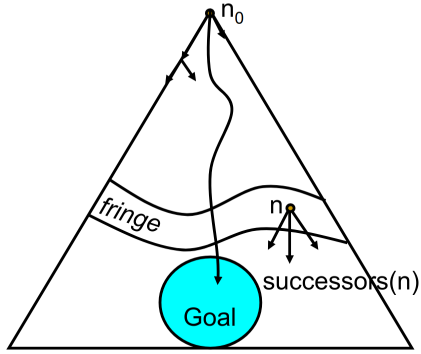
- Maze solver
- Lecturer assignment problem
- Student class picking problem
- Traveling salesman problem (TSP)



MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY
8:00h ACTIVITY	8:00h ACTIVITY	8:00h ACTIVITY	8:00h ACTIVITY	8:00h ACTIVITY
9:00h ACTIVITY	9:00h ACTIVITY	9:00h ACTIVITY	9:00h ACTIVITY	9:00h ACTIVITY
10:00h ACTIVITY	10:00h ACTIVITY	10:00h ACTIVITY	10:00h ACTIVITY	10:00h ACTIVITY

Outline

Search tree



Search tree:

- Represent the branching paths through a **state graph**.
- Usually **much** larger than the state graph.
- Can a finite state graph give an infinite search tree?

Tree search algorithms

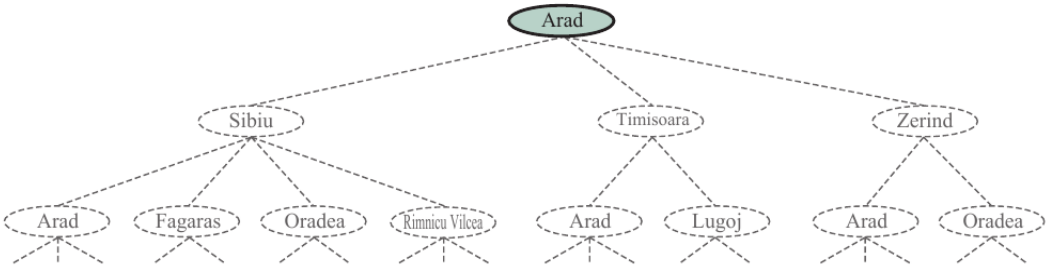
Basic idea:

- offline, simulated exploration of state space by generating successors of already-explored states

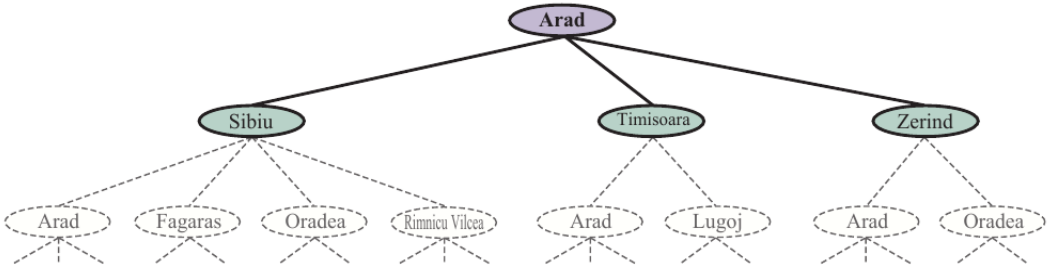
```
function TREE-SEARCH(problem, strategy) returns a solution, or failure
    initialize the search tree using the initial state of problem
    loop do
        if there are no candidates for expansion then return failure
        choose a leaf node for expansion according to strategy
        if the node contains a goal state then return the corresponding solution
        else expand the node and add the resulting nodes to the search tree
```

Example: the Romania problem

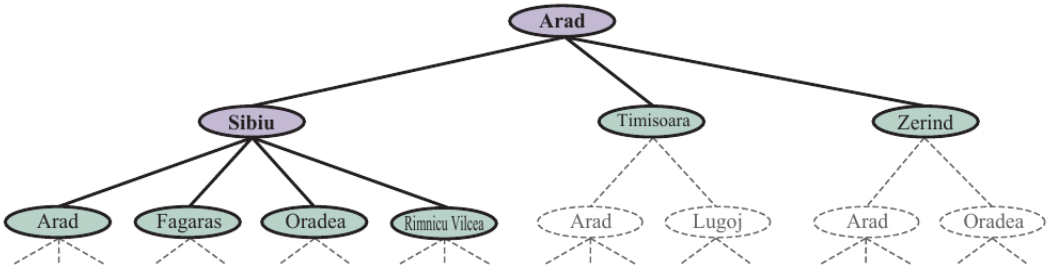
Three partial search trees for finding a route from Arad to Bucharest. Nodes that have been *expanded* are lavender with bold letters; nodes on the frontier that have been *generated* but not yet *expanded* are in green; the set of states corresponding to these two types of nodes are said to have been reached. Nodes that could be generated next are shown in faint dashed lines.



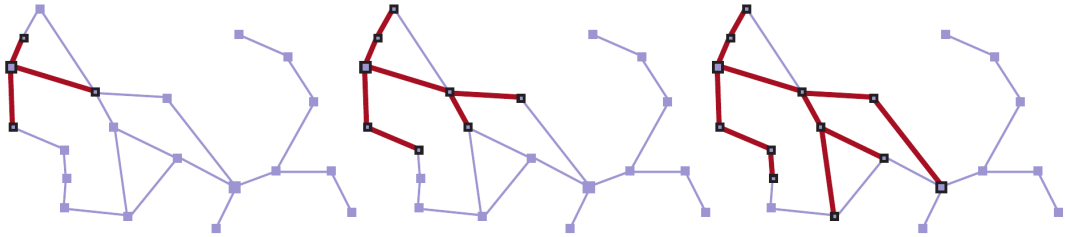
Example: the Romania problem



Example: the Romania problem



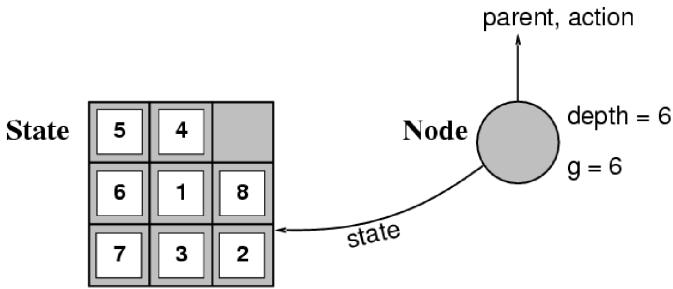
Example: the Romania problem



- A sequence of search trees generated by a graph search on the Romania problem. At each stage, we have expanded every node on the frontier, extending every path with all applicable actions that don't result in a state that has already been reached.
- Notice that at the third stage, the topmost city (Oradea) has two successors, both of which have already been reached by other paths, so no paths are extended from Oradea.

Implementation: states vs. nodes

- A state is a (representation of) a physical configuration
- A node is a data structure constituting part of a search tree includes state, parent node, action, path cost $g(x)$, depth



- The `Expand` function creates new nodes, filling in the various fields and using the `SuccessorFn` of the problem to create the corresponding states.

Search strategies

A search strategy is defined by picking the order of node expansion
Strategies are evaluated along the following dimensions:

- **Completeness:** Is the algorithm guaranteed to find a solution when there is one, and to correctly report failure when there is not?
- **Cost optimality:** Does it find a solution with the lowest path cost of all solutions?
- **Time complexity:** How long does it take to find a solution? This can be measured in seconds, or more abstractly by the number of states and actions considered.
- **Space complexity:** How much memory is needed to perform the search.

In the next lecture

Will discuss on uninformed search strategies (DFS, BFS, etc.).