Artificial Intelligence
Uninformed Search

Michael McConnell

University of Vermont

March 15, 2022

Problem Solving Agent

```
function SIMPLE-PROBLEM-SOLVING-AGENT(percept) returns an action
   static: seq, an action sequence, initially empty
           state, some description of the current world state
           goal, a goal, initially null
           problem, a problem formulation

   state ← UPDATE-STATE(state, percept)
   if seq is empty then
       goal ← FORMULATE-GOAL(state)
       problem ← FORMULATE-PROBLEM(state, goal)
       seq ← SEARCH(problem)
   action ← RECOMMENDATION(seq, state)
   seq ← REMAINDER(seq, state)
   return action
```

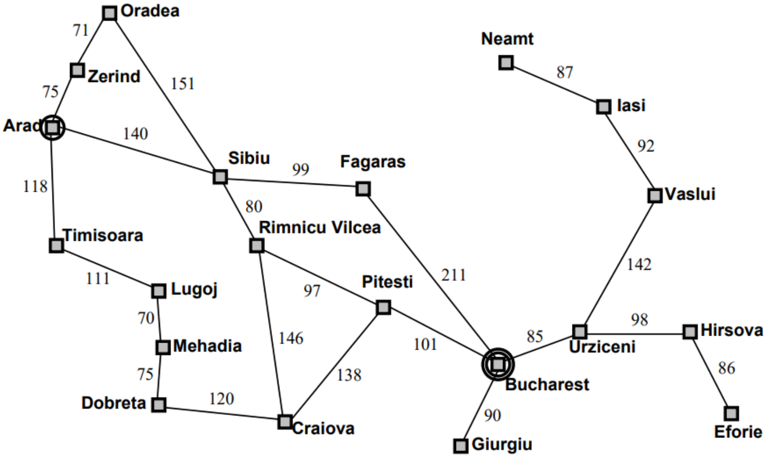
[1]

[1] Artificial Intelligence: A Modern Approach, Norvig and Russell, 2020

Types of Problems

- **Deterministic**: Fully observable single state problems. These involve an agent knowing exactly which state it will be in given a certain action, each solution is a sequence of actions.
- **Non-Observable**: These are problems wherein an agent may not have any idea about where it actually is at any given point in time. The solution still takes the form of a sequence.
- **Nondeterministic**: These are also called partially observable problems. In these problem types, each observation provides new information about the current state. The solution is a kind of plan or overarching policy to be followed. Searching and execution of action are often combined.
- **Unknown State Spaces**: What we generally refer to as Online Search, or problems involving exploration of a search space.

Map of Romania



[2]

[2] Artificial Intelligence: A Modern Approach, Norvig and Russell, 2020

Map Example

- **What is our Goal?**
- We want to arrive in Bucharest
- We are starting out from Arad.
- Each state is one of the cities.
- Each action is a drive between the cities.
- The solution consists of a sequence of the cities.

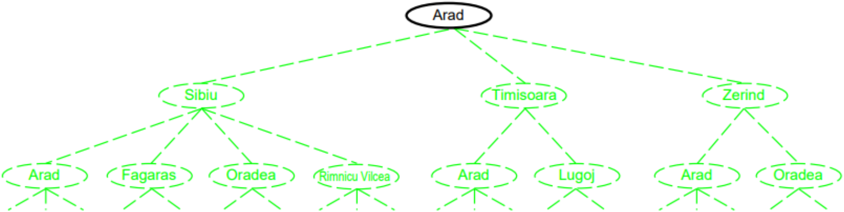
Tree Search Algorithm Outline

```
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
    end
```

[3]

[3] Artificial Intelligence: A Modern Approach, Norvig and Russell, 2020

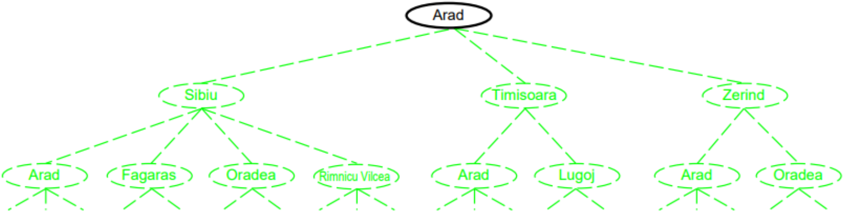
- **Essentially we will simulate the exploration of the state space and generate successors of already-explored states (it just expands the tree).**



4

[4] Artificial Intelligence: A Modern Approach, Norvig and Russell, 2020

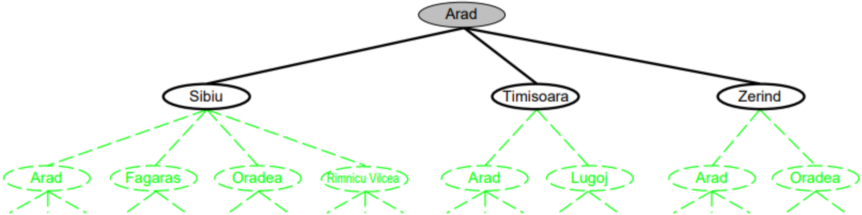
Map Example



5

[5] Artificial Intelligence: A Modern Approach, Norvig and Russell, 2020

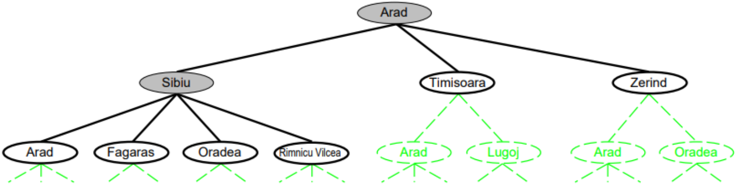
Map Example



6

[6] Artificial Intelligence: A Modern Approach, Norvig and Russell, 2020

Map Example



[7]

[7] Artificial Intelligence: A Modern Approach, Norvig and Russell, 2020

Tree Search Expansion Algorithm

```
function TREE-SEARCH( problem, fringe) returns a solution, or failure
    fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
    loop do
        if fringe is empty then return failure
        node ← REMOVE-FRONT(fringe)
        if GOAL-TEST(problem, STATE(node)) then return node
        fringe ← INSERTALL(EXPAND(node, problem), fringe)
```

```
function EXPAND( node, problem) returns a set of nodes
    successors ← the empty set
    for each action, result in SUCCESSOR-FN(problem, STATE[node]) do
        s ← a new NODE
        PARENT-NODE[s] ← node;  ACTION[s] ← action;  STATE[s] ← result
        PATH-COST[s] ← PATH-COST[node] + STEP-COST(node, action, s)
        DEPTH[s] ← DEPTH[node] + 1
        add s to successors
    return successors
```

[8]

[8] Artificial Intelligence: A Modern Approach, Norvig and Russell, 2020

What is a State vs a Node

- **States:** A representation of the physical configuration.
- **Node:** A kind of data structure or object which holds information about the search tree. In our example this includes parent nodes, children, depth, path-cost, etc.
- **Expand:** The function creates new nodes, filling the various fields up
- **SuccessorFN**: This function is gathering all the corresponding states according to our problem description.

Search Strategy

- A strategy is defined by picking the order of each expansion. Expansion is really the core of our actions here.
- **Strategies are evaluated along four dimensions**:
  - Completeness → Does the algorithm always find a solution if one exists?
  - Time Complexity → What number of nodes are required to be examined or 'expanded' in the process?
  - Space Complexity → How many nodes do we have stored in memory throughout the process?
  - Optimality → Does it always find a least-cost solution? (this implies actions have costs, sometimes they might not)

Time and Space Complexity

- **We will be measuring these in terms of:**
- **b**: The maximum branching factor of the search tree
- **d**: The depth of the least-cost solution
- **m**: The maximum depth of the state space (which can potentially be infinite).

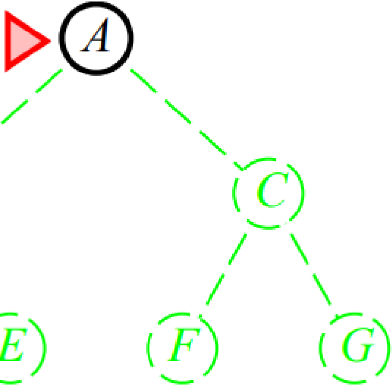
Uninformed Search Strategies

- When a strategy is uninformed it adopts a fixed rule for selecting what to expand next.
- The rule for what to do never changes, it has no regard for the search problem that is being solved.
- These strategies cannot utilize any domain specific information about the search problem that is being solved.

Breadth First Search

- With BFS we define expansion as simply taking the shallowest unexpanded node and expanding it.
- Here the frontier or 'fringe' is defined as a FIFO queue, where all the new successors get placed at the end each time.

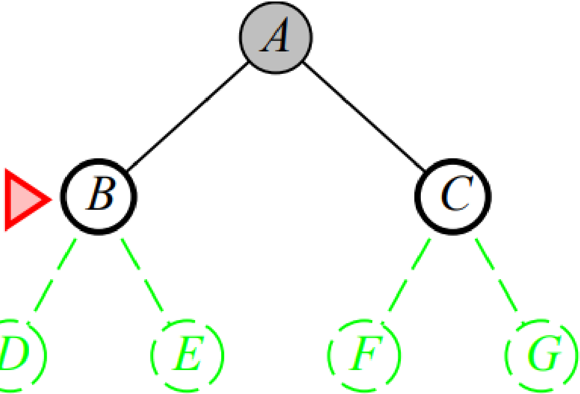
Breadth First Search Example



[9]

[9] Artificial Intelligence: A Modern Approach, Norvig and Russell, 2020

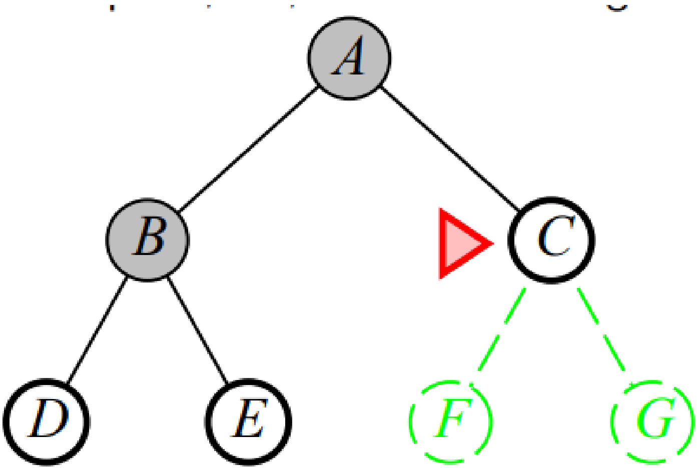
Breadth First Search Example



[10]

[10] Artificial Intelligence: A Modern Approach, Norvig and Russell, 2020

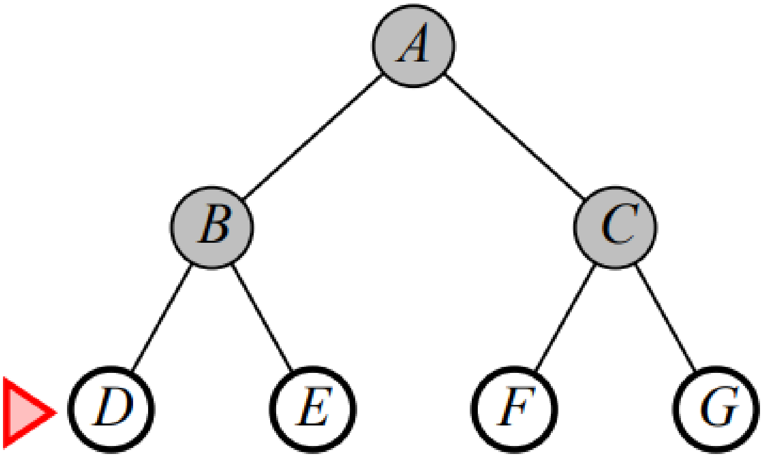
Breadth First Search Example



11

[11] Artificial Intelligence: A Modern Approach, Norvig and Russell, 2020

Breadth First Search Example



12

[12] Artificial Intelligence: A Modern Approach, Norvig and Russell, 2020

Breadth First Search Properties

Completeness

- All shorter paths are epanded prior to any longer path, thus we eventually examine every path at each depth. If a solution exists at that depth it is found.
- This is complete as long as the b (The maximum branching factor of the search tree) is finite.

Breadth First Search Properties

Time Complexity

- We can define this as $O(b^{d+1})$. Meaning that it is exponential with respect to the depth of the least-cost solution.
- Reminder: b is the maximum branching factor, and d is the depth of the shortest solution.
- $1 + b + b^2 + b^3 + .. + b^d + b(b^d - 1)$

Breadth First Search Properties
Space Complexity

- This is going to be the same as the last slide $O(b^{d+1})$.

Breadth First Search Properties

Optimality

- If path costs are random like in our Romania example, no. Yes, if the cost for each step is 1.
- Why?: The 'shortest' solution on the tree is not necessarily the cheapest solution if actions have varying costs.

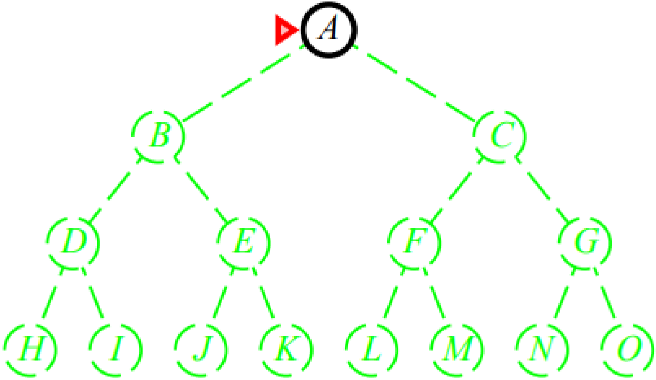
Uniform Cost Search

- This is equivalent to breadth first search, but with all the step costs equal.
- The frontier is a queue ordered by the path cost of each node with the lowest first.
- **Complete**: Yes if there is a step cost $\geq \epsilon > 0$

Depth First Search

- With DFS we define expansion as simply taking the deepest unexpanded node and expanding it.
- The frontier here is defined as a LIFO queue, where all the new successors get placed at the front each round.

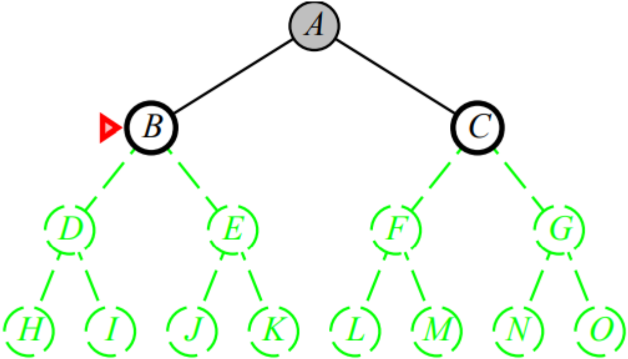
Depth First Search Example



13

[13] Artificial Intelligence: A Modern Approach, Norvig and Russell, 2020

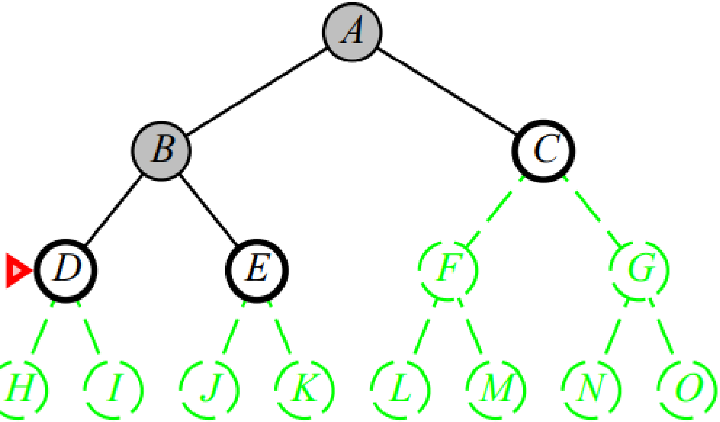
Depth First Search Example



[14]

[14] Artificial Intelligence: A Modern Approach, Norvig and Russell, 2020

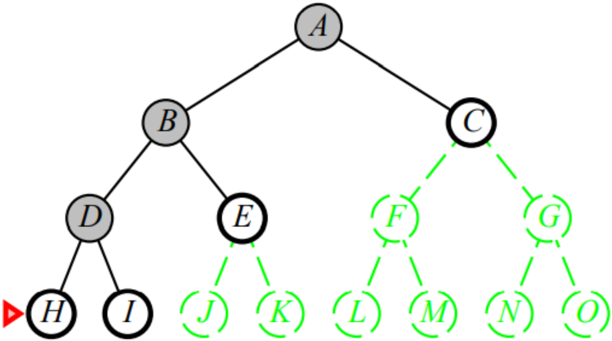
Depth First Search Example



15

[15] Artificial Intelligence: A Modern Approach, Norvig and Russell, 2020

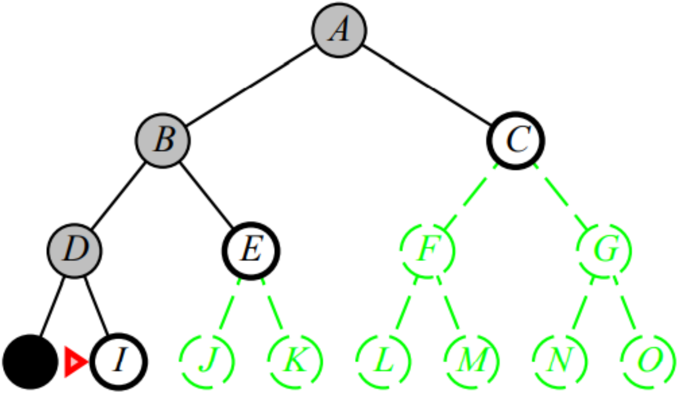
Depth First Search Example



16

[16] Artificial Intelligence: A Modern Approach, Norvig and Russell, 2020

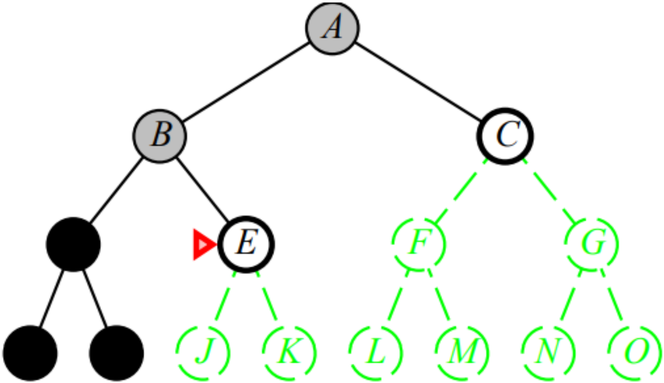
Depth First Search Example



17

[17] Artificial Intelligence: A Modern Approach, Norvig and Russell, 2020

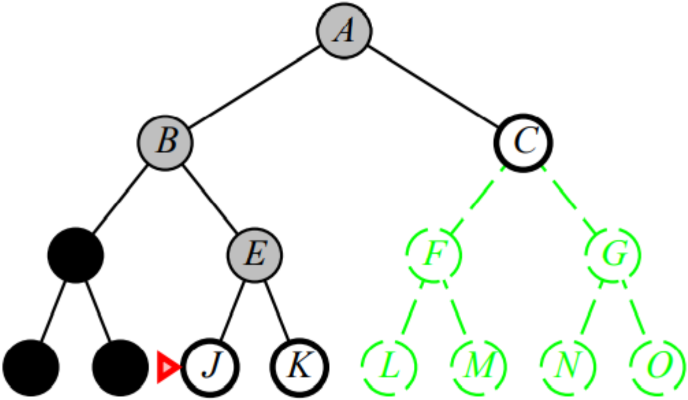
Depth First Search Example



18

[18] Artificial Intelligence: A Modern Approach, Norvig and Russell, 2020

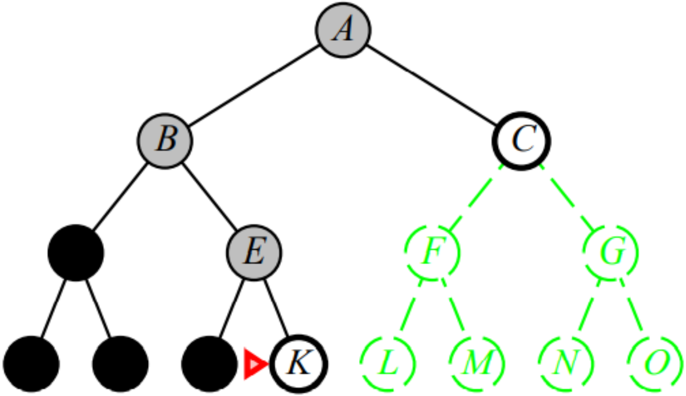
Depth First Search Example



[19]

[19] Artificial Intelligence: A Modern Approach, Norvig and Russell, 2020

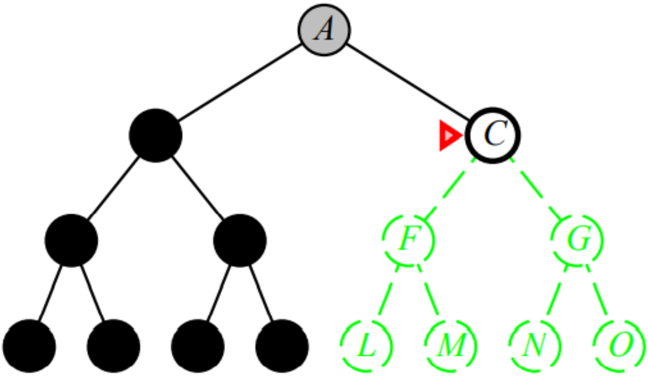
Depth First Search Example



[20]

[20] Artificial Intelligence: A Modern Approach, Norvig and Russell, 2020

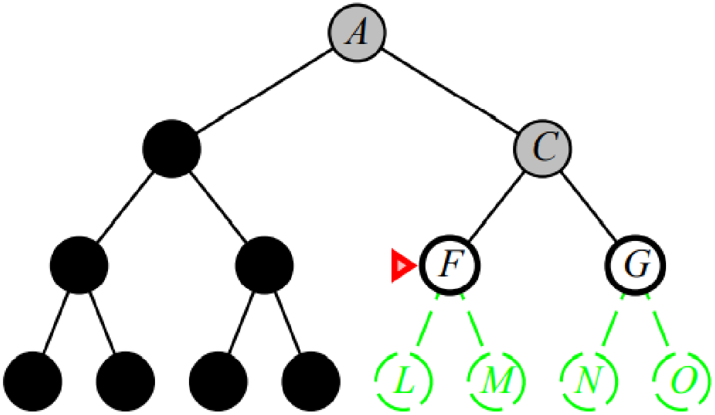
Depth First Search Example



21

[21] Artificial Intelligence: A Modern Approach, Norvig and Russell, 2020

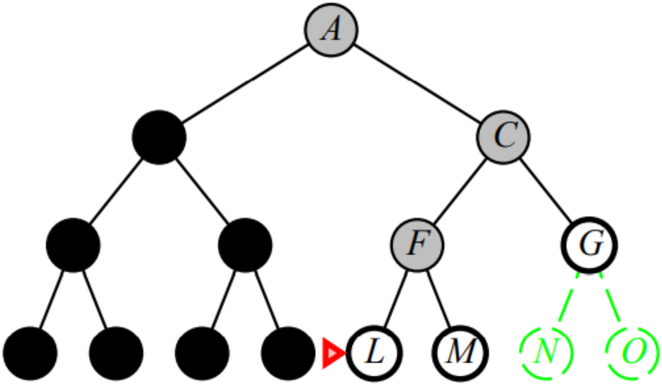
Depth First Search Example



22

[22] Artificial Intelligence: A Modern Approach, Norvig and Russell, 2020

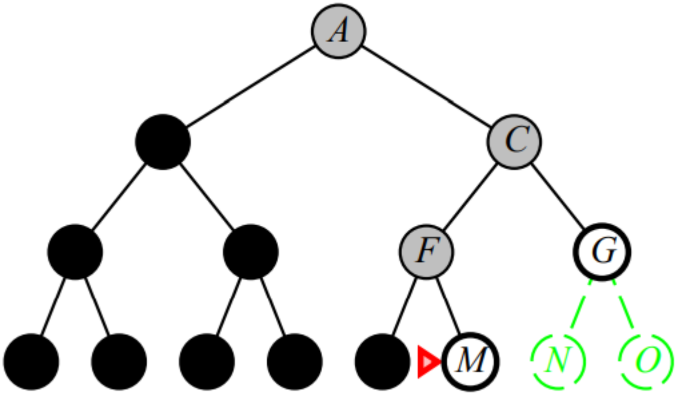
Depth First Search Example



23

[23] Artificial Intelligence: A Modern Approach, Norvig and Russell, 2020

Depth First Search Example



24

[24] Artificial Intelligence: A Modern Approach, Norvig and Russell, 2020

Depth First Search Properties

Completeness

- No, this fails for any infinite depth spaces or spaces with loops. It can be modified to avoid repeating states along a given pathway.
- If you modify it, it can be complete in the context of a finite spaces.

Depth First Search Properties

Time Complexity

- $O(b^m)$ so this gives us a bad time complexity in any case where the maximum depth is much larger than depth of the least-cost solution.
- If the solutions are dense it may be faster than breadth-first.

Depth First Search Properties

Space Complexity

- **Space Complexity:** $O(bm)$

Depth First Search Properties

Optimality

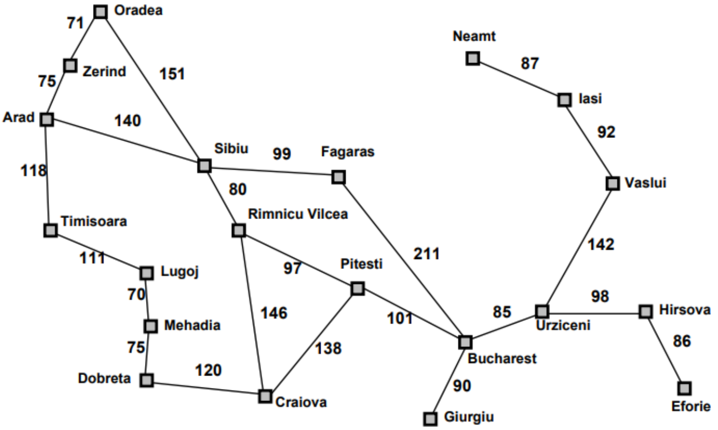
- **Optimality: No.**

Informed Search

- Good search strategies are defined by picking the **order of node expansion**
- The core idea of BFS algorithms is that we can use some evaluation function as a criteria. We evaluate each node according to some heuristic estimate of desirability'.
- **Expansion:** Expand te most desirable unexpanded node.
- The frontier is defined as a queue sorted in decreasing order of our desirability metric.
- The two common types are Greedy and A*.

- In our Romania example, we as outside observers never want to be going in the opposite direction from the goal. This is clearly completely wrong.
- We want to instead look-ahead to the goal and try to orient ourselves towards that.
- In the textbook these heuristic functions are often written h(n), which is taken to mean the estimated cost of the cheapest path from a node in to a goal node.

Straight-Line Heuristic



Straight−line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

[25]

[25] Artificial Intelligence: A Modern Approach, Norvig and Russell, 2020

Greedy Search

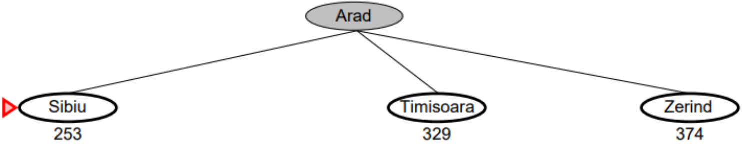
- **Evaluation Function:** We are defining the evaluation function h(n) to be the straight line distance from a node n to Bucharest.
- The greedy search simply expands the node that appears based on our heuristic to be closest to the goal.

Greedy Search Example

26

[26] Artificial Intelligence: A Modern Approach, Norvig and Russell, 2020

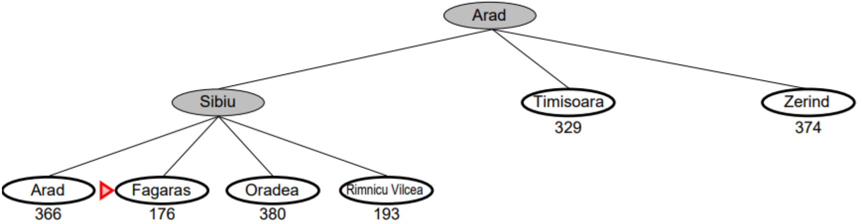
Greedy Search Example



27

[27] Artificial Intelligence: A Modern Approach, Norvig and Russell, 2020

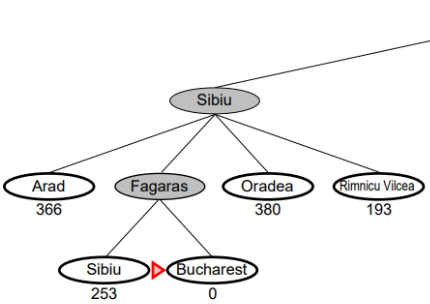
Greedy Search Example



28

[28] Artificial Intelligence: A Modern Approach, Norvig and Russell, 2020

Greedy Search Example



29

[29] Artificial Intelligence: A Modern Approach, Norvig and Russell, 2020

Greedy Search Properties

Completeness

- No, it can get stuck in a loop, for example in the Romania map. If Oradea is the goal we can get stuck in a loop Iasi $\Rightarrow$ Neamt $\Rightarrow$ Iasi $\Rightarrow$ Neamt, etc. (show on the graph if people are curious)
- It is complete in a finite space with checks for repeating states.

Greedy Search Properties

Time Complexity

- $O(b^m)$ but in cases where a good heuristic is developed the time complexity can be reduced significantly.

Greedy Search Properties

Space Complexity

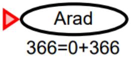
- $O(b^m)$ **because it has to keep all nodes and their associated desirability metric in memory as it goes.**

- **Optimality: No.**

A* Search Algorithm

- $f(n) = g(n) + h(n)$
- $g(n)$: the cost so far to reach n
- $h(n)$: the cost to the goal from n
- $f(n)$: estimated total cost of path through n to the goal.

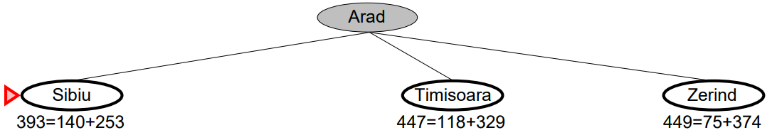
A* Search Example



30

[30] Artificial Intelligence: A Modern Approach, Norvig and Russell, 2020

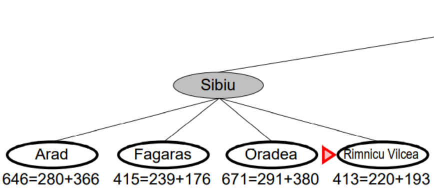
A* Search Example



[31]

[31] Artificial Intelligence: A Modern Approach, Norvig and Russell, 2020

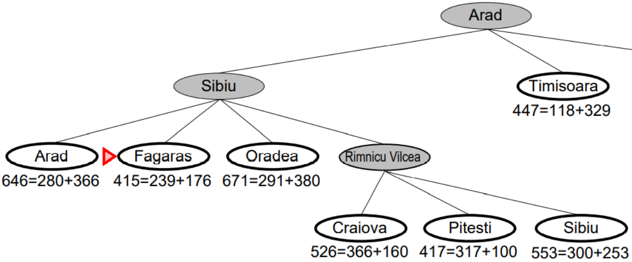
A* Search Example



32

[32] Artificial Intelligence: A Modern Approach, Norvig and Russell, 2020

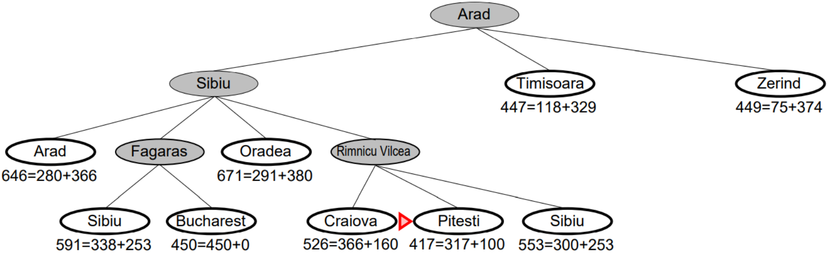
A* Search Example



33

[33] Artificial Intelligence: A Modern Approach, Norvig and Russell, 2020

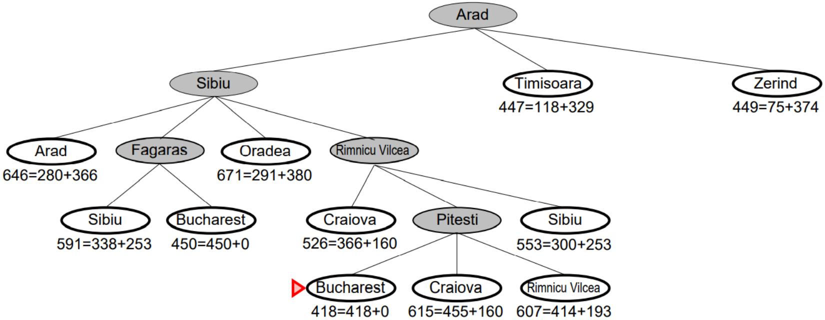
A* Search Example



34

34 Artificial Intelligence: A Modern Approach, Norvig and Russell, 2020

A* Search Example



35

[35] Artificial Intelligence: A Modern Approach, Norvig and Russell, 2020