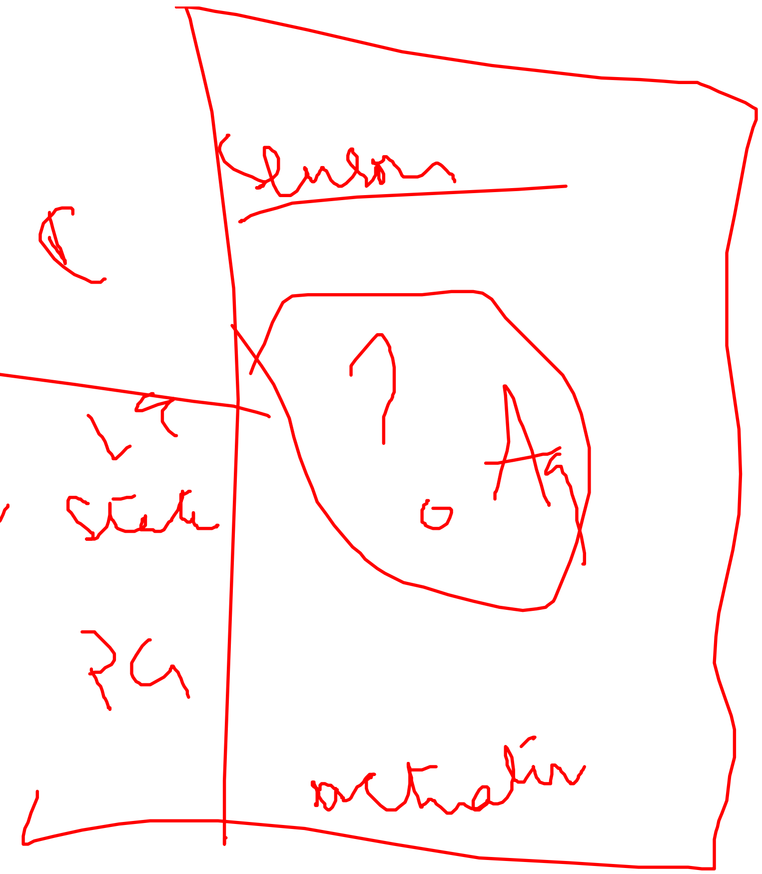
ARTIFICIAL INTELLIGENCE

REVISION

- simple reflex agents
- reflex agents with state
- goal-based agents
- utility-based agents

All of these 4 types can be turned into learning agents



PROBLEM SOLVING

Problem Solving Agents -> Atomic Representation

Planning Agents -> Factored or Structured Representation

Assumption: Solution of a problem is a fixed sequence of actions and this sequence DOES NOT depend upon future percepts

(Chapter 4 relaxes above mentioned assumption)

Prev_3 lec
Foundation of AI
- AI?
- Agents?
- Env

Strategies
AI

INFORMED VS UNINFORMED SEARCH

uninformed search algorithms—algorithms that are given no information about the problem other than its definition.

Informed search algorithms, on the other hand, can do quite well given some guidance on where to look for solutions

PROBLEM SOLVING AGENT

4 — Tasks

1) Goal Formulation
2) Problem Formulation
3) Searching Solution
4) Executing Solution ×

1- GOAL FORMULATION

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

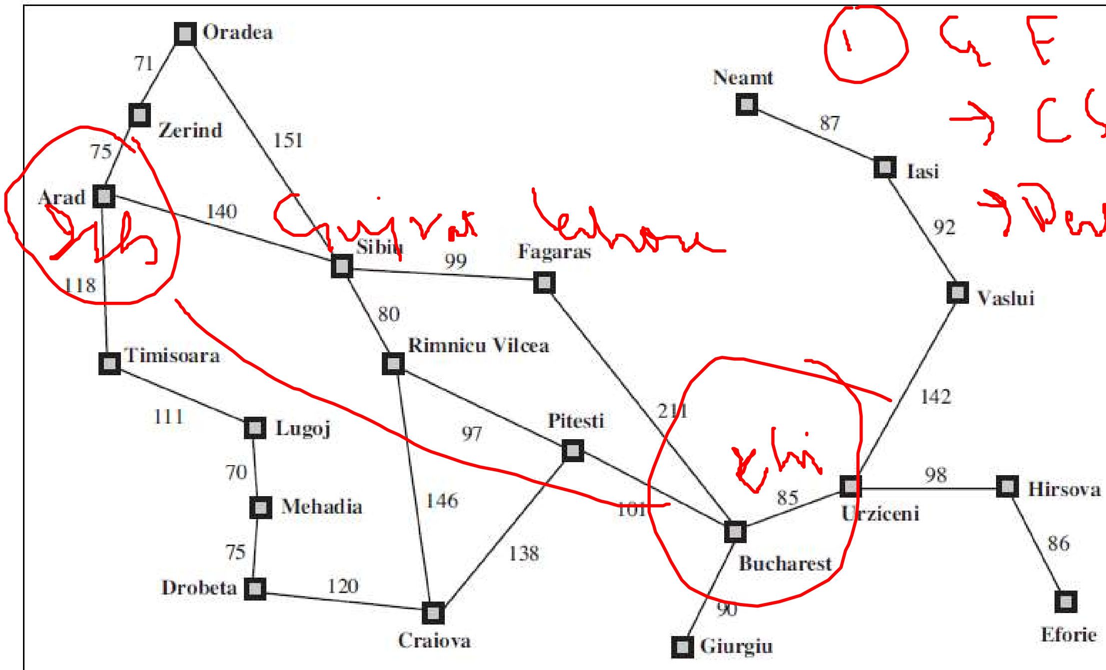


Figure 3.2 A simplified road map of part of Romania.

Performance Measure: Visit as many cities as possible, spend as low as possible on fuel.

Current State: In Arad → A set of states in which goal is satisfied

Possible Goal: In Bucharest

Prob Solv

2- PROBLEM FORMULATION

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

1 2

The agent will consider actions at the level of driving from one major town to another. Each state therefore corresponds to being in a particular town.

Vacuum

— Move

Go Left, Go Right, Go forward, Go reverse. If these 4 actions are considered agent will never go out of the parking lot let alone reaching Bucharest.

Define

— listing down possible actions

Possible Action: Go Cityname

STATE SPACE GRAPH

env

On the right is the state space graph of our problem

State Space - the set of all states reachable from the initial state by any sequence of actions.

all possibilities

A **path** in the state space is a sequence of states connected by a sequence of actions.

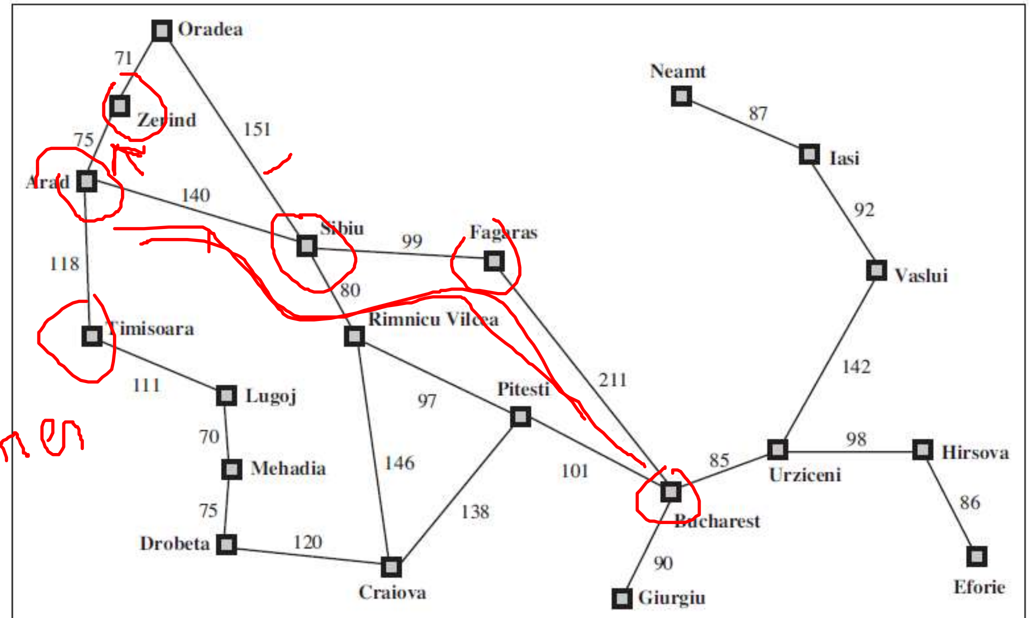
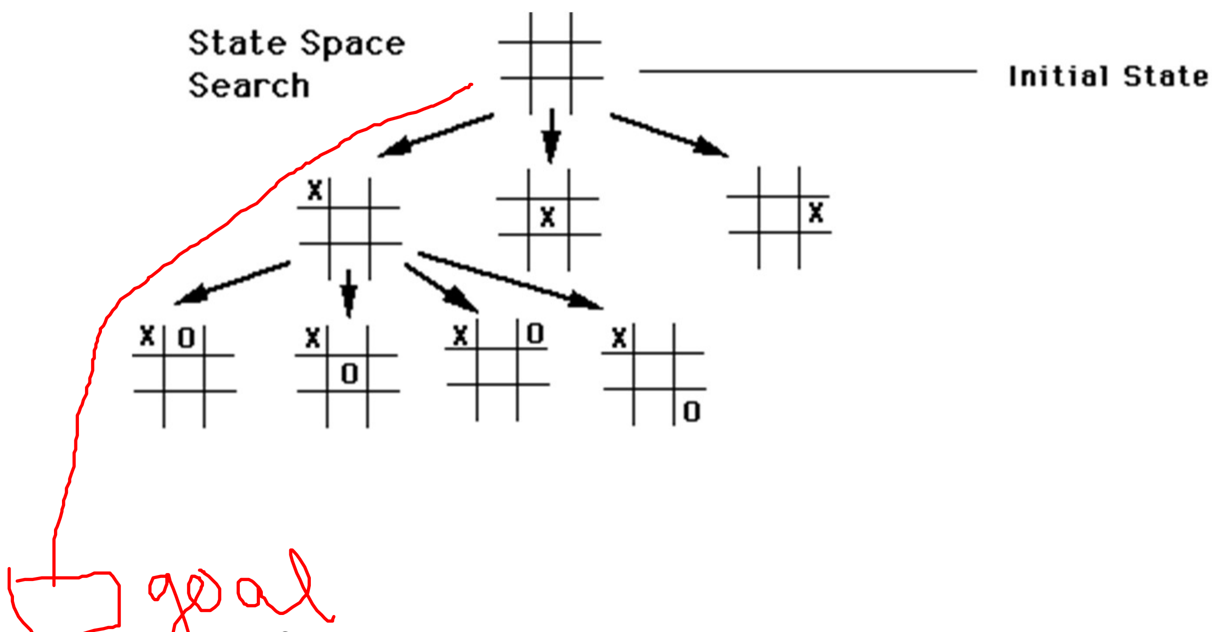


Figure 3.2 A simplified road map of part of Romania.

Rep of env = State

Agent can be in any one of these state

STATE SPACE GRAPH



How would a state space of chess look like?

TOY PROBLEM: 8 PUZZLE

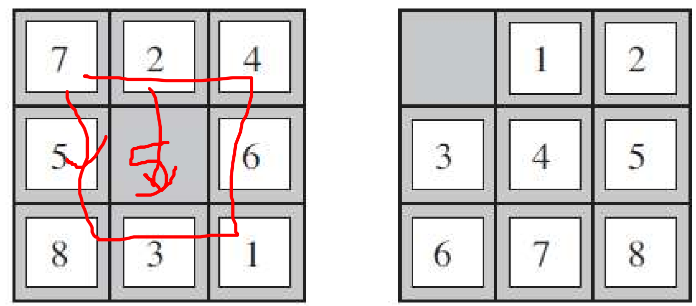
Any state can be used as starting and Goal states.

Actions: *Left*, *Right*, *Up*, or *Down*.

Transition model: Given a state and action, this returns the resulting state; for example, if we apply *Left* to the start state in adjacent figure, the resulting state has the 5 and the blank switched.

consequence of taking an action

Path Cost: Each step costs 1, so the path cost is the number of steps in the path.

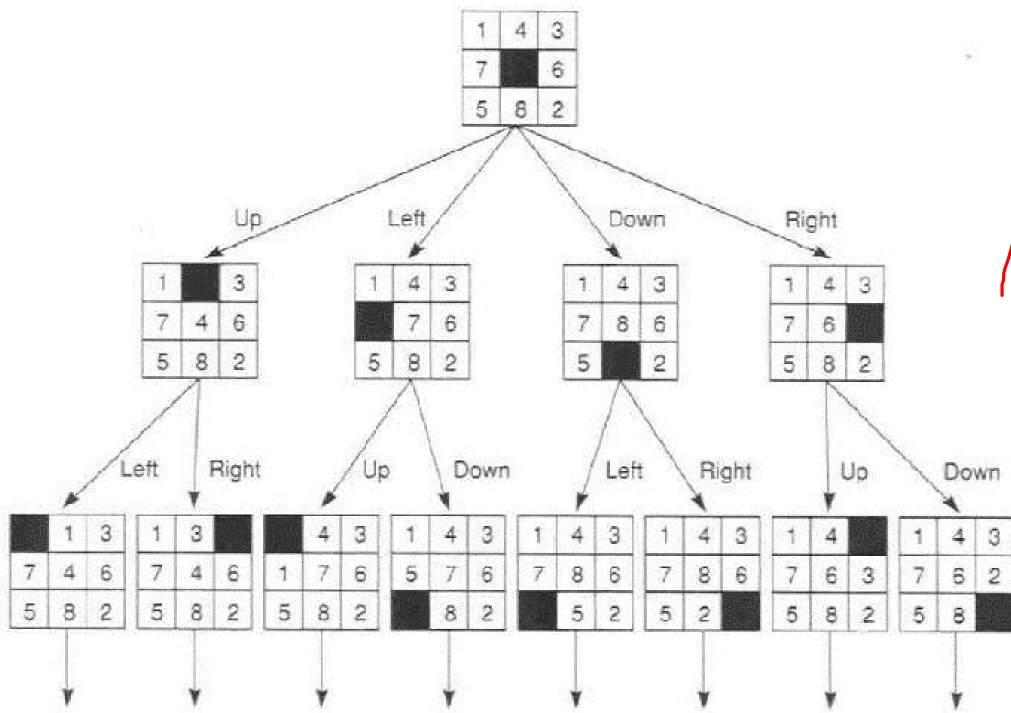


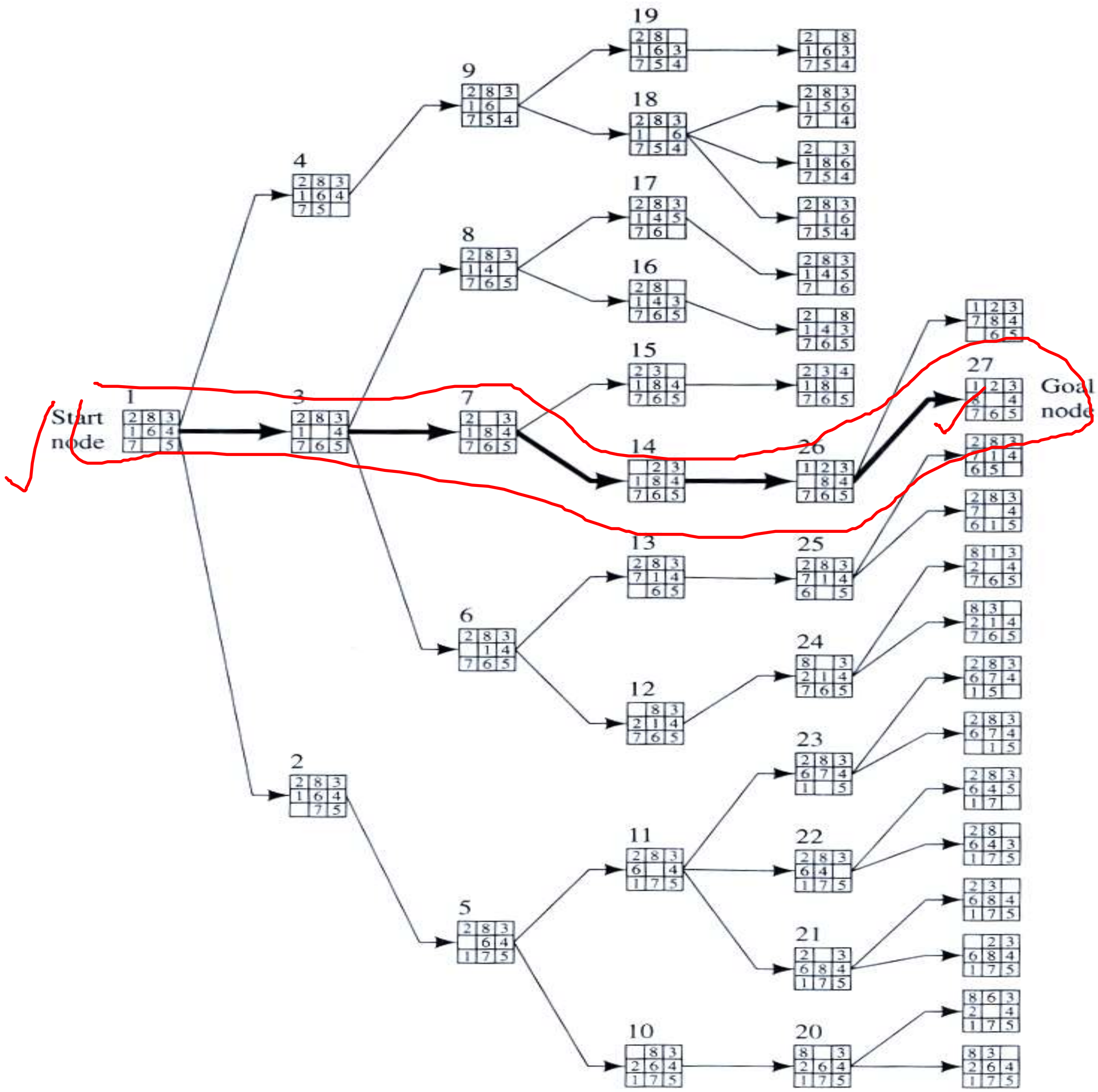
Start State

Goal State

Not fixed

TOY PROBLEM: 8 PUZZLE





Solve the game

→ Obj of my AI agent was to solve the game.

PROBLEM DEFINITION

A problem consists of five components,

- Initial State
- Possible Actions
- Transition Model
- Goal Test
- Path Cost

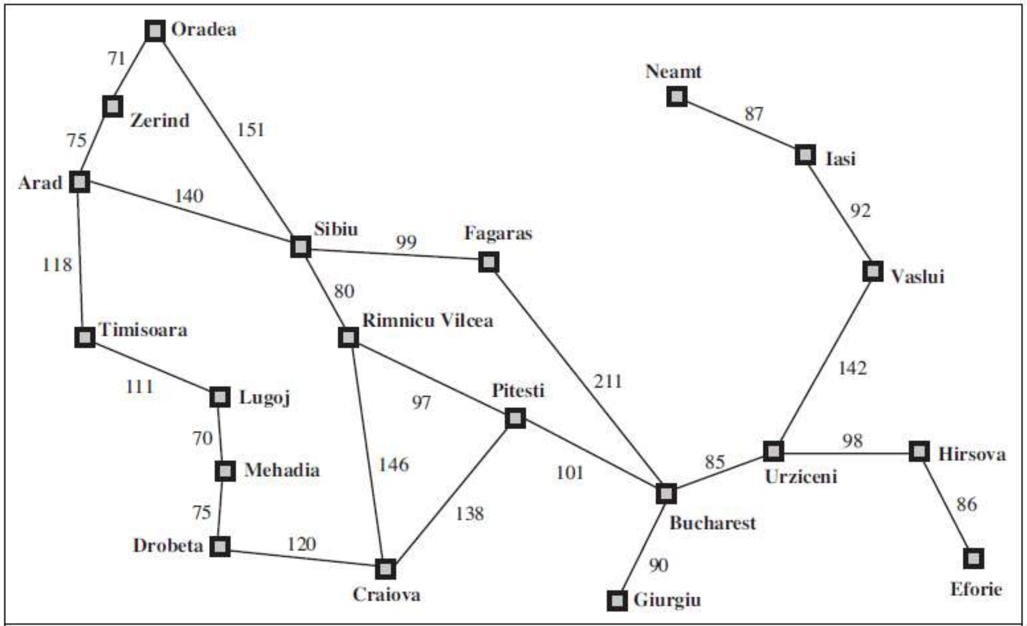


Figure 3.2 A simplified road map of part of Romania.

PROBLEM DEFINITION

A problem consists of five components,

- Initial State : in (Arad)
- Possible Actions (given a state): e.g., from Arad possible actions are {Go(Sibiu), Go(Timisoara), Go(Zerind)}.
- Transition Model: specifies the relationship between a state, a possible action and the resulting successor state e.g.,

  RESULT(In(Arad),Go(Zerind)) = In(Zerind)

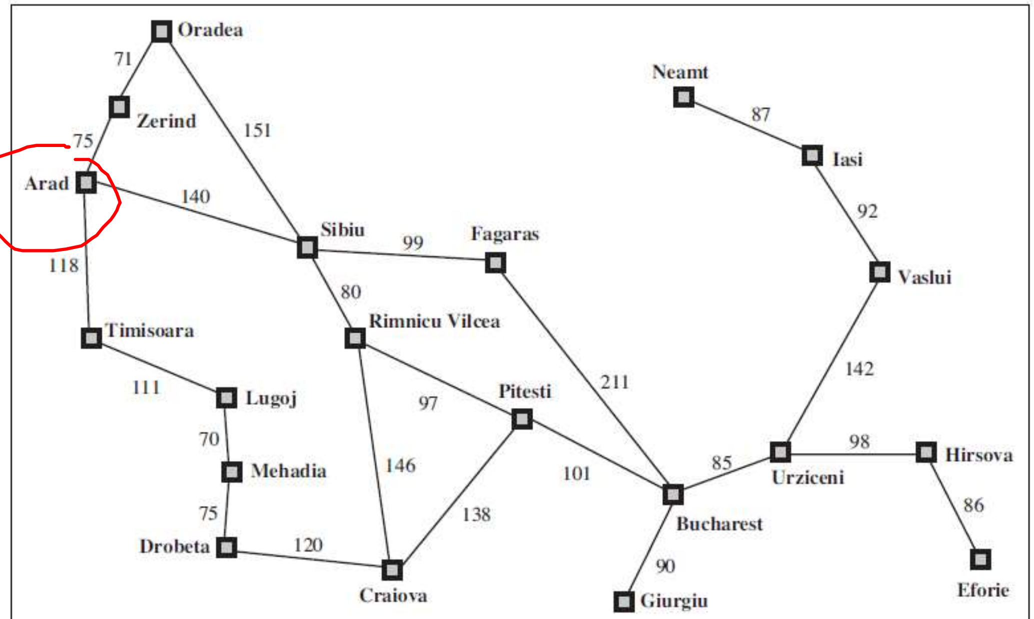


Figure 3.2 A simplified road map of part of Romania.

PROBLEM DEFINITION

A problem consists of five components,

- Initial State
- Possible Actions
- Transition Model
- Goal Test (whether a given state is a goal state) e.g. {In(Bucharest)}.
- Path Cost (based upon agent's performance measure). Two paths shown on the right. Step cost (the cost of a single action within a path)

Goal formulation

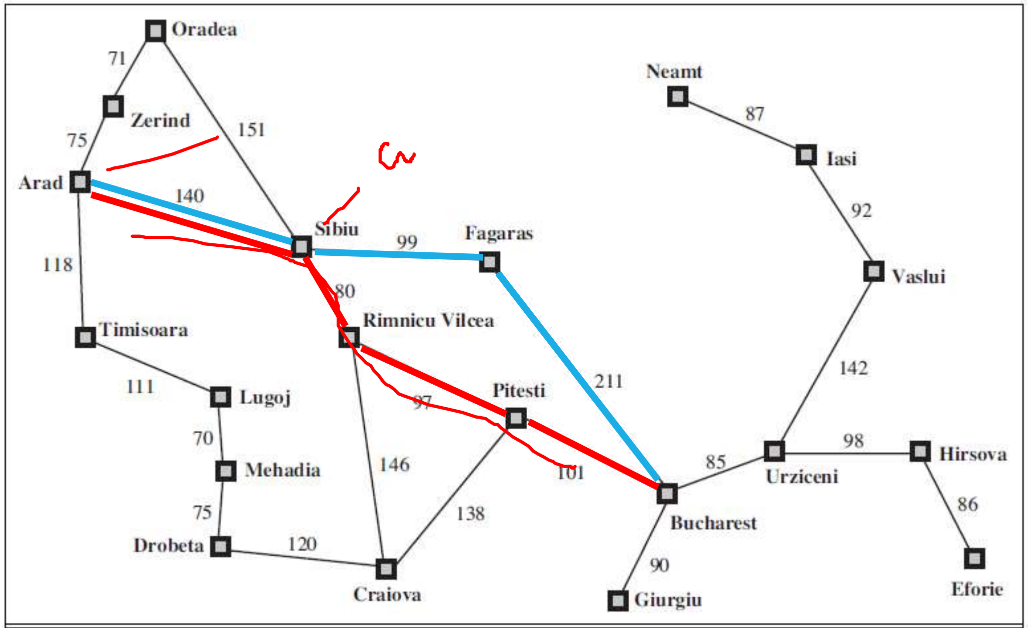


Figure 3.2 A simplified road map of part of Romania.

SOLUTION DEFINITION

A **solution** to a problem is an action sequence that leads from the initial state to a goal state.

Solution quality is measured by the path cost function, and an **optimal solution** has the lowest path cost among all solutions.

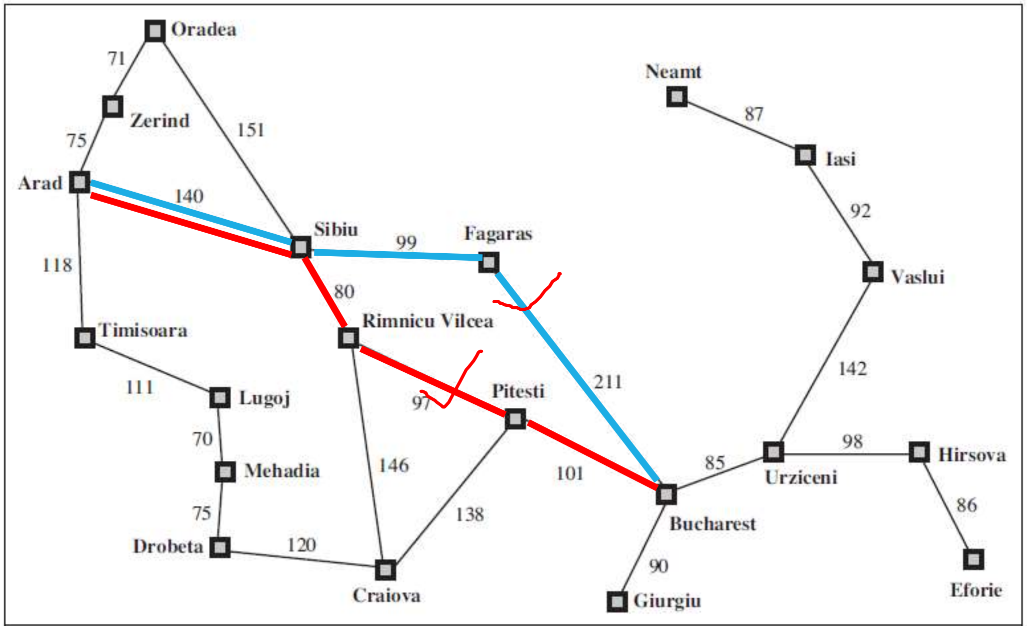


Figure 3.2 A simplified road map of part of Romania.

TREE SEARCH

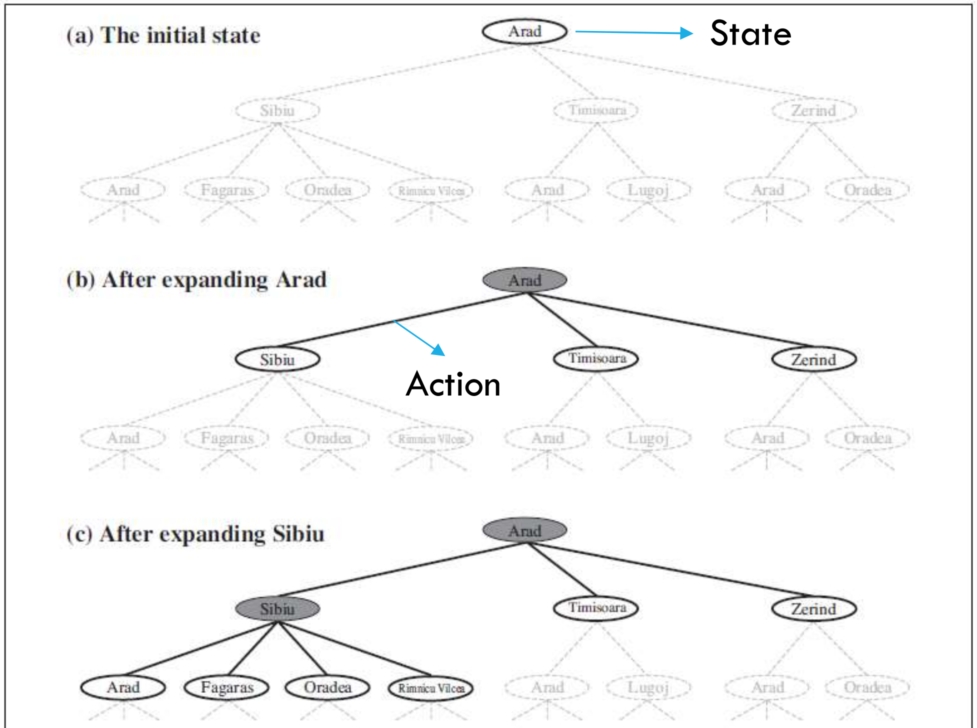


Figure 3.6 Partial search trees for finding a route from Arad to Bucharest. Nodes that have been expanded are shaded; nodes that have been generated but not yet expanded are outlined in bold; nodes that have not yet been generated are shown in faint dashed lines.

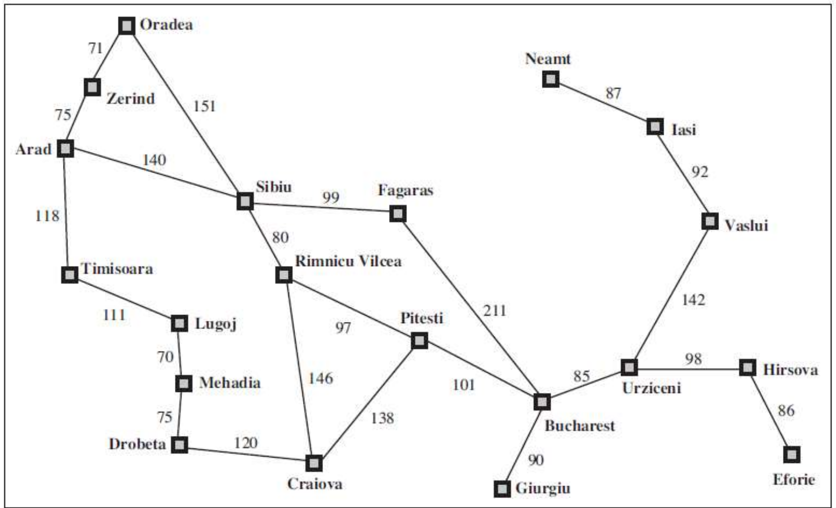
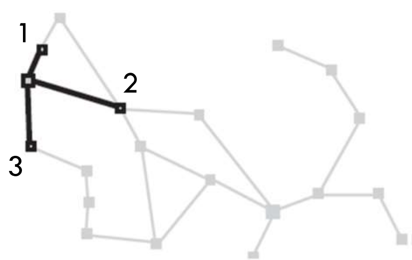


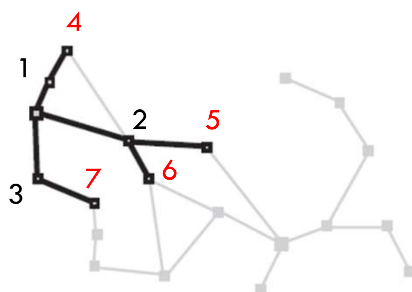
Figure 3.2 A simplified road map of part of Romania.

GRAPH SEARCH

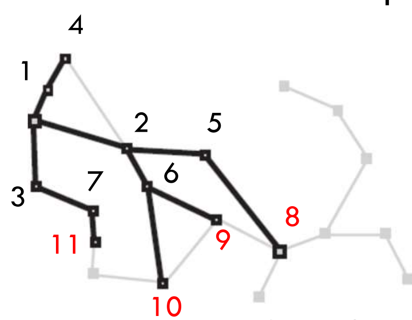
Use an explore set / closed list / frontier to remember the states already visited



Step 1



Step 2



Step 3

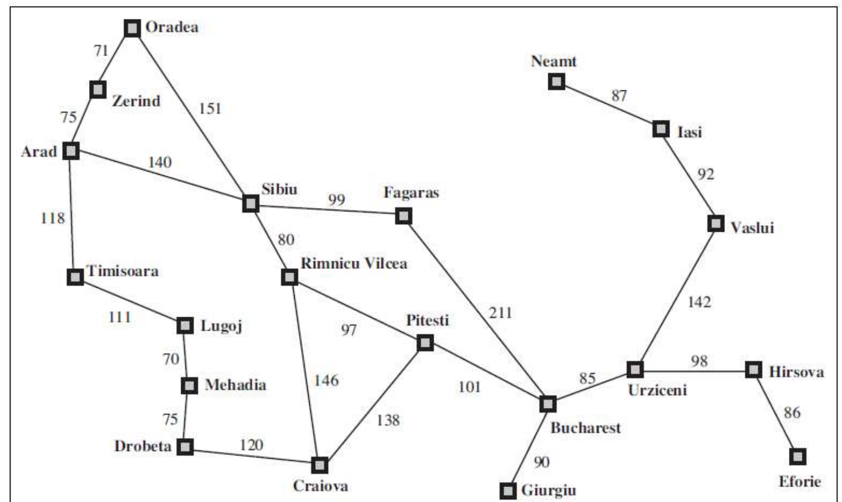


Figure 3.2 A simplified road map of part of Romania.

INFRASTRUCTURE FOR SEARCH ALGORITHMS

- n.STATE: the state in the state space to which the node corresponds;
- n.PARENT: the node in the search tree that generated this node;
- n.ACTION: the action that was applied to the parent to generate the node;
- n.PATH-COST: the cost, traditionally denoted by $g(n)$, of the path from the initial state to the node, as indicated by the parent pointers.

Frontier / expanded nodes / unexpanded nodes are stored in the form of a queue (can be FIFO, LIFO (stack) or priority queue)

MEASURING PROBLEM-SOLVING PERFORMANCE

Completeness: Is the algorithm guaranteed to find a solution when there is one?

Optimality: Does the strategy find the optimal solution.

Time complexity: How long does it take to find a solution?

Space complexity: How much memory is needed to perform the search?

MEASURING PROBLEM-SOLVING PERFORMANCE

Complexity in a tree or a graph is measured by:

1)b, the **branching factor** or maximum number of children of any node

2)d, the **depth** of the goal node (i.e., the number of steps along the path from the root)

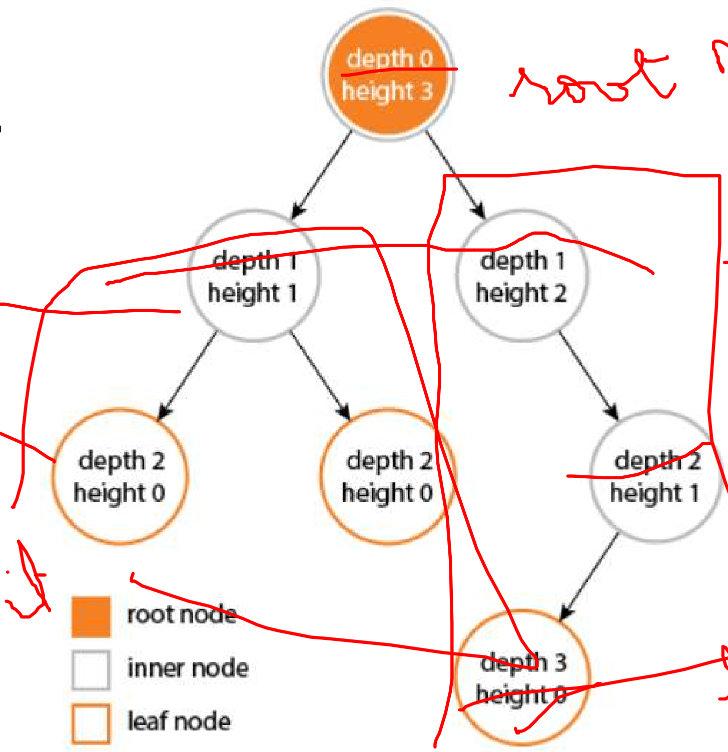
3)m, the maximum possible depth of entire tree.

Time is often measured in terms of the number of nodes generated during the search.

Space in terms of the maximum number of nodes stored in memory.

MEASURING PROBLEM-SOLVING PERFORMANCE

A binary tree has a branching factor



root D = 0

level

leaf

H

no of levels below it

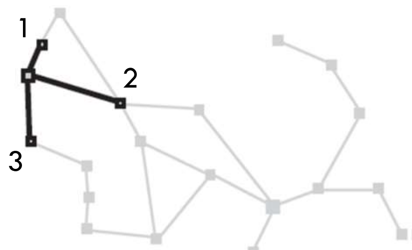
level 2

level 3

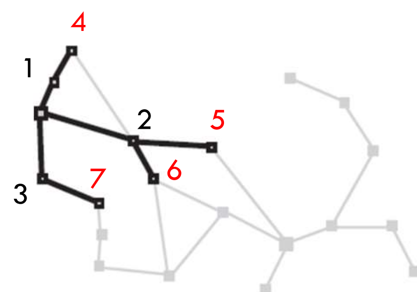
COMPLEXITY

Branching Factor, b=3 (Max num of children for any node is 3)

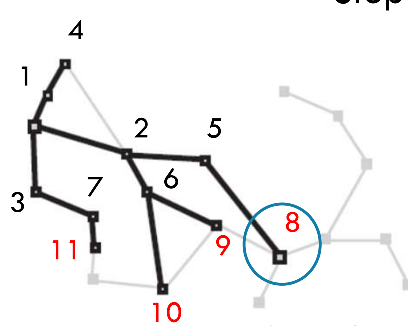
Depth, d = 3 (Bucharest is 3 steps away from Arad)



Step 1



Step 2



Step 3

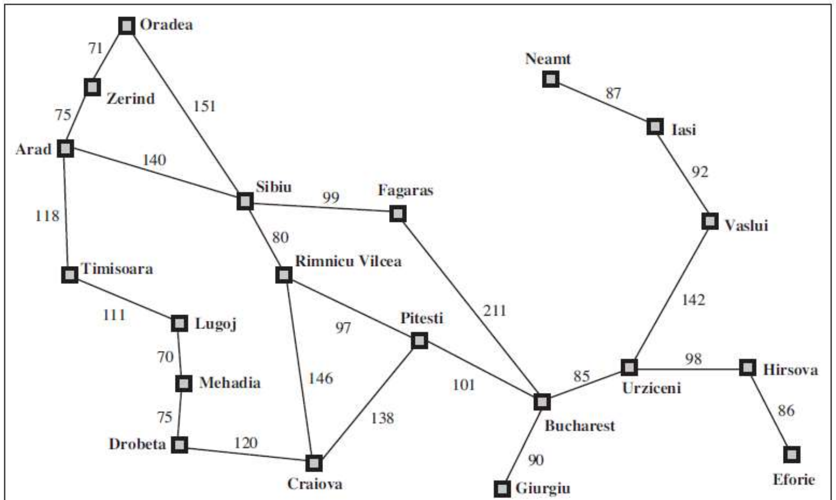


Figure 3.2 A simplified road map of part of Romania.

INFORMED VS UNINFORMED SEARCH

uninformed search algorithms—algorithms that are given no information about the problem other than its definition.

- Breadth First Search
- Uniform Cost Search
- Depth First Search
- Depth Limited Search
- Iterative Deepening Depth First Search
- Bi Directional Search

Informed search algorithms, on the other hand, can do quite well given some guidance on where to look for solutions

Summary:

→ AI Agents → ~~Solve~~ Solve

Prob Solv

1. informed
2. Uninformed

1. — G ← (S, P)
2. — P ← (S, A)
3. — S. Sear (algo)

↓

BFS

Define

- Initial State
- Action poss
- Transition Mod
- G. Test
- P Cost

BREADTH FIRST SEARCH

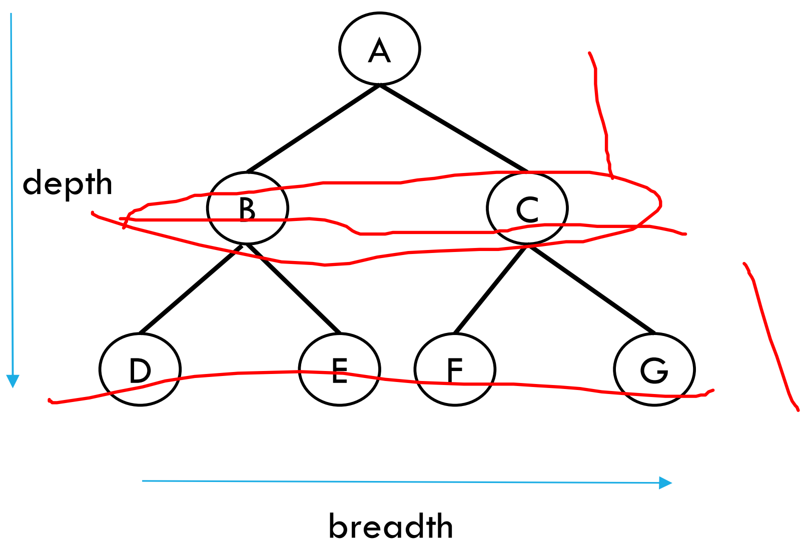
Expand Shallowest Node First

Frontier (or fringe): nodes in queue to be explored

Reached: Nodes that are already explored

Frontier is a first-in-first-out (FIFO) queue, i.e., new successors go at end of the queue.

Goal-Test when inserted



BREADTH FIRST SEARCH

Expand Shallowest Node First

Frontier (or fringe): nodes in queue to be explored

Reached: Nodes that are already explored

Frontier is a first-in-first-out (FIFO) queue, i.e., new successors go at end of the queue.

Goal-Test when inserted

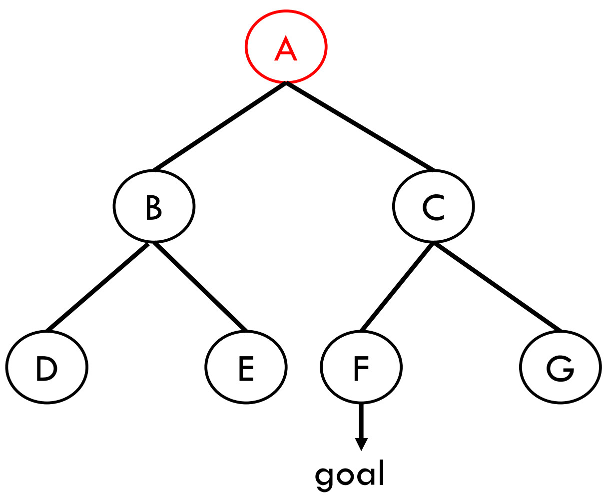
```
function BREADTH-FIRST-SEARCH(problem) returns a solution node or failure
  node ← NODE(problem.INITIAL)
  if problem.IS-GOAL(node.STATE) then return node
  frontier ← a FIFO queue, with node as an element
  reached ← {problem.INITIAL}
  while not IS-EMPTY(frontier) do
    node ← POP(frontier)
    for each child in EXPAND(problem, node) do
      s ← child.STATE
      if problem.IS-GOAL(s) then return child
      if s is not in reached then
        add s to reached
        add child to frontier
  return failure
```

BREADTH FIRST SEARCH

Is A a goal state?

Frontier = [A]

Reached = [ ]

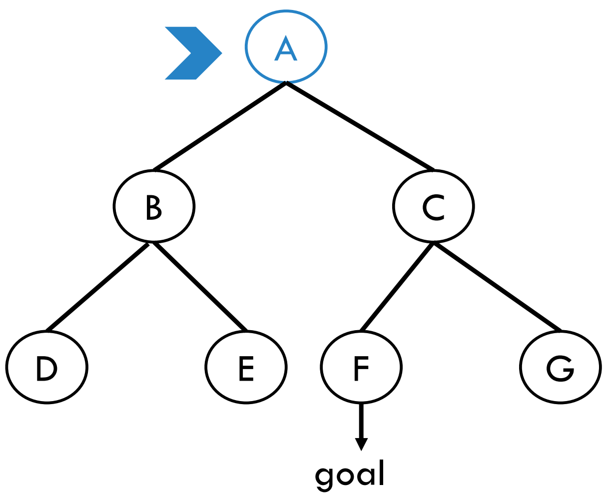



```
function BREADTH-FIRST-SEARCH(problem) returns a solution node or failure
  node ← NODE(problem.INITIAL)
  if problem.IS-GOAL(node.STATE) then return node
  frontier ← a FIFO queue, with node as an element
  reached ← {problem.INITIAL}
  while not IS-EMPTY(frontier) do
    node ← POP(frontier)
    for each child in EXPAND(problem, node) do
      s ← child.STATE
      if problem.IS-GOAL(s) then return child
      if s is not in reached then
        add s to reached
        add child to frontier
  return failure
```

BREADTH FIRST SEARCH

Frontier = [A]
Reached = [ ]

Frontier = [ ]
Reached = [A]




```
function BREADTH-FIRST-SEARCH(problem) returns a solution node or failure
  node ← NODE(problem.INITIAL)
  if problem.IS-GOAL(node.STATE) then return node
  frontier ← a FIFO queue, with node as an element
  reached ← {problem.INITIAL}
  while not IS-EMPTY(frontier) do
    node ← POP(frontier)
    for each child in EXPAND(problem, node) do
      s ← child.STATE
      if problem.IS-GOAL(s) then return child
      if s is not in reached then
        add s to reached
        add child to frontier
  return failure
```

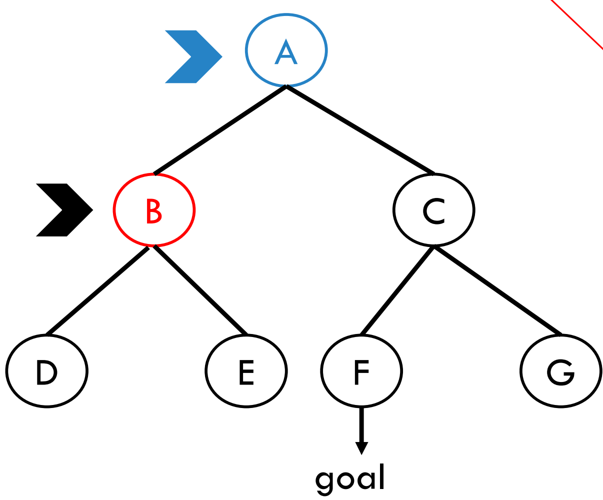
Node

Child

BREADTH FIRST SEARCH

Frontier = [ ]
Reached = [A]

Frontier = [B]
Reached = [A]




```
function BREADTH-FIRST-SEARCH(problem) returns a solution node or failure
  node ← NODE(problem.INITIAL)
  if problem.IS-GOAL(node.STATE) then return node
  frontier ← a FIFO queue, with node as an element
  reached ← {problem.INITIAL}
  while not IS-EMPTY(frontier) do
    node ← POP(frontier)
    for each child in EXPAND(problem, node) do
      s ← child.STATE
      if problem.IS-GOAL(s) then return child
      if s is not in reached then
        add s to reached
        add child to frontier
  return failure
```

Is B a goal state?

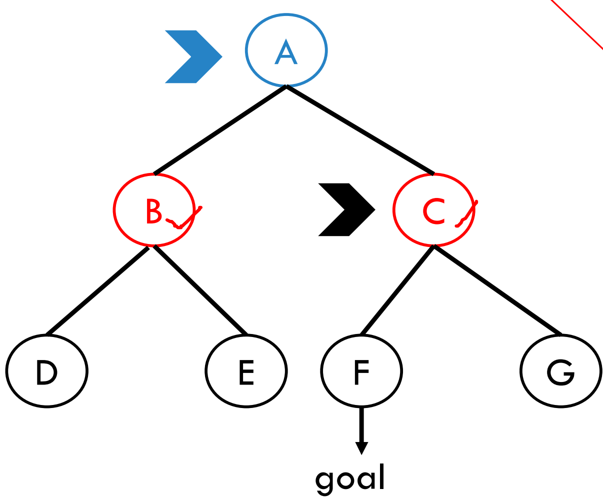
Node

Child

BREADTH FIRST SEARCH

Frontier = [B]
Reached = [A]

Frontier = [B, C]
Reached = [A]



Is C a goal state?

```
function BREADTH-FIRST-SEARCH(problem) returns a solution node or failure
  node ← NODE(problem.INITIAL)
  if problem.IS-GOAL(node.STATE) then return node
  frontier ← a FIFO queue, with node as an element
  reached ← {problem.INITIAL}
  while not IS-EMPTY(frontier) do
    node ← POP(frontier)
    for each child in EXPAND(problem, node) do
      s ← child.STATE
      if problem.IS-GOAL(s) then return child
      if s is not in reached then
        add s to reached
        add child to frontier
  return failure
```

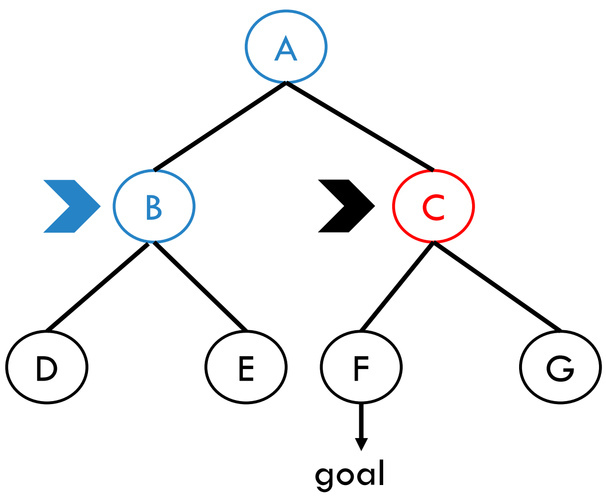
Node

Child

BREADTH FIRST SEARCH

Frontier = [B, C]
Reached = [A]

Frontier = [C]
Reached = [A, B]




```
function BREADTH-FIRST-SEARCH(problem) returns a solution node or failure
  node ← NODE(problem.INITIAL)
  if problem.IS-GOAL(node.STATE) then return node
  frontier ← a FIFO queue, with node as an element
  reached ← {problem.INITIAL}
  while not IS-EMPTY(frontier) do
    node ← POP(frontier)
    for each child in EXPAND(problem, node) do
      s ← child.STATE
      if problem.IS-GOAL(s) then return child
      if s is not in reached then
        add s to reached
        add child to frontier
  return failure
```

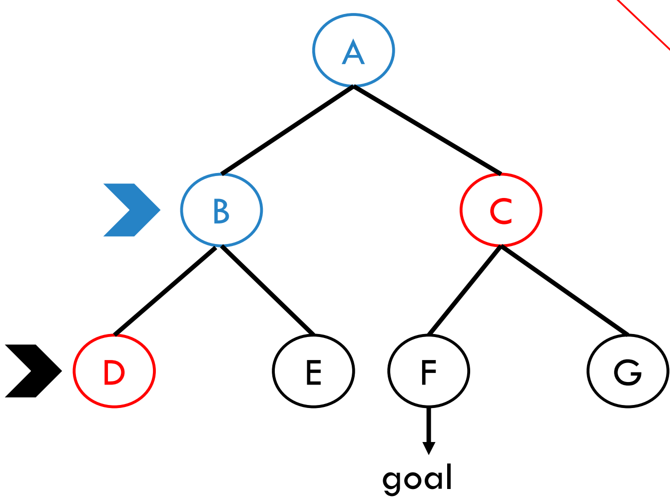
Node

Child

BREADTH FIRST SEARCH

Frontier = [C]
Reached = [A, B]

Frontier = [C, D]
Reached = [A, B]




```
function BREADTH-FIRST-SEARCH(problem) returns a solution node or failure
  node ← NODE(problem.INITIAL)
  if problem.IS-GOAL(node.STATE) then return node
  frontier ← a FIFO queue, with node as an element
  reached ← {problem.INITIAL}
  while not IS-EMPTY(frontier) do
    node ← POP(frontier)
    for each child in EXPAND(problem, node) do
      s ← child.STATE
      if problem.IS-GOAL(s) then return child
      if s is not in reached then
        add s to reached
        add child to frontier
  return failure
```

Is D a goal state?

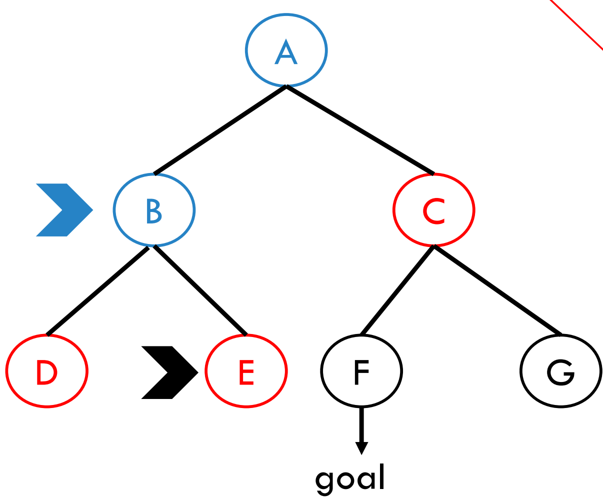
Node

Child

BREADTH FIRST SEARCH

Frontier = [C, D]
Reached = [A, B]

Frontier = [C, D, E]
Reached = [A, B]




```
function BREADTH-FIRST-SEARCH(problem) returns a solution node or failure
  node ← NODE(problem.INITIAL)
  if problem.IS-GOAL(node.STATE) then return node
  frontier ← a FIFO queue, with node as an element
  reached ← {problem.INITIAL}
  while not IS-EMPTY(frontier) do
    node ← POP(frontier)
    for each child in EXPAND(problem, node) do
      s ← child.STATE
      if problem.IS-GOAL(s) then return child
      if s is not in reached then
        add s to reached
        add child to frontier
  return failure
```

Is E a goal state?

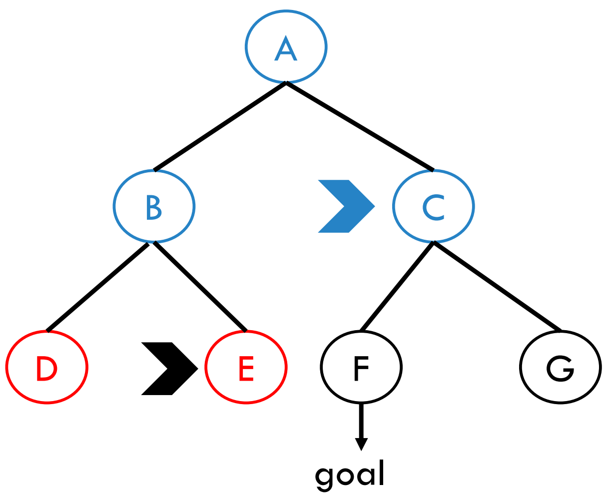
Node

Child

BREADTH FIRST SEARCH

Frontier = [C, D, E]
Reached = [A, B]

Frontier = [D, E]
Reached = [A, B, C]




```
function BREADTH-FIRST-SEARCH(problem) returns a solution node or failure
  node ← NODE(problem.INITIAL)
  if problem.IS-GOAL(node.STATE) then return node
  frontier ← a FIFO queue, with node as an element
  reached ← {problem.INITIAL}
  while not IS-EMPTY(frontier) do
    node ← POP(frontier)
    for each child in EXPAND(problem, node) do
      s ← child.STATE
      if problem.IS-GOAL(s) then return child
      if s is not in reached then
        add s to reached
        add child to frontier
  return failure
```

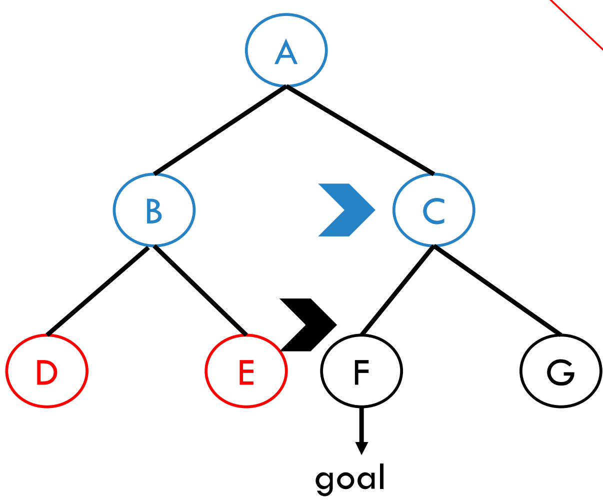
Node

Child

BREADTH FIRST SEARCH

Frontier = [D, E]
Reached = [A, B, C]

Frontier = [D, E]
Reached = [A, B, C]



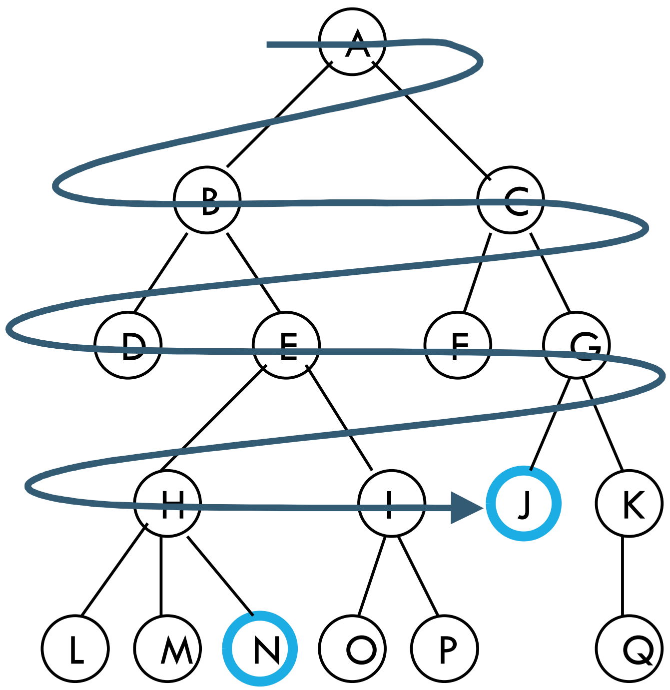

```
function BREADTH-FIRST-SEARCH(problem) returns a solution node or failure
  node ← NODE(problem.INITIAL)
  if problem.IS-GOAL(node.STATE) then return node
  frontier ← a FIFO queue, with node as an element
  reached ← {problem.INITIAL}
  while not IS-EMPTY(frontier) do
    node ← POP(frontier)
    for each child in EXPAND(problem, node) do
      s ← child.STATE
      if problem.IS-GOAL(s) then return child
      if s is not in reached then
        add s to reached
        add child to frontier
  return failure
```

Is F a goal state?

Node

Child

BFS: SUMMARY



① 2 ③ FIFO

② 1 3 F / E

④ G T

G tasks ?

return

Add to Frontier

① F → E

→ Add child

BFS: COMPLETENESS

Completeness: Yes (guaranteed to find a solution if there exists one)

if the shallowest goal node is at some finite depth d, breadth-first search will eventually find it after generating all shallower nodes (provided the branching factor b is finite) .

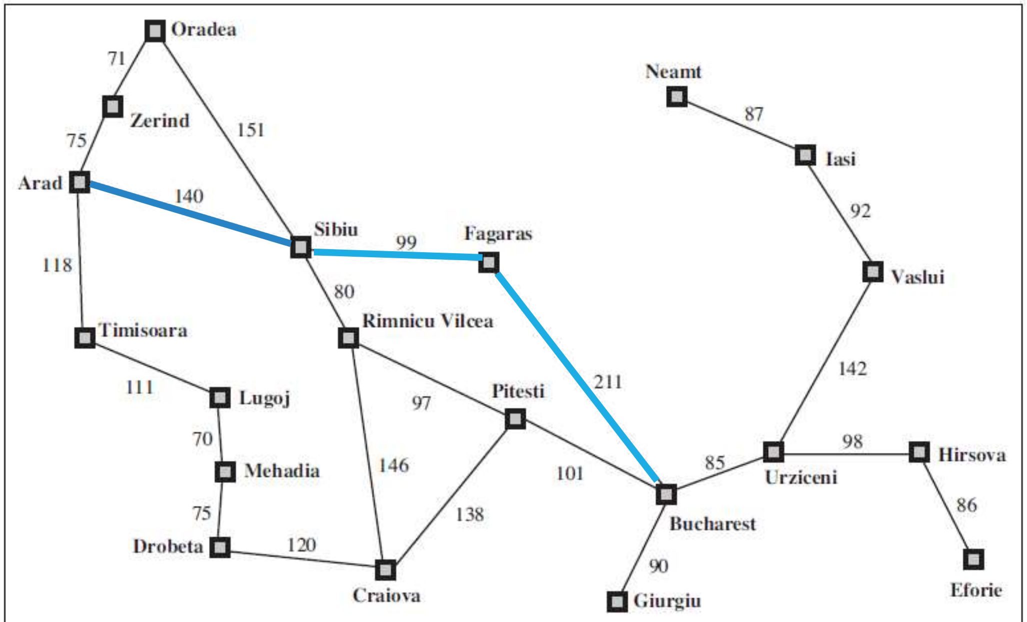


Figure 3.2 A simplified road map of part of Romania.

BFS goal path

BFS: OPTIMAL?

Not necessarily optimal

Only optimal if every action has same cost.

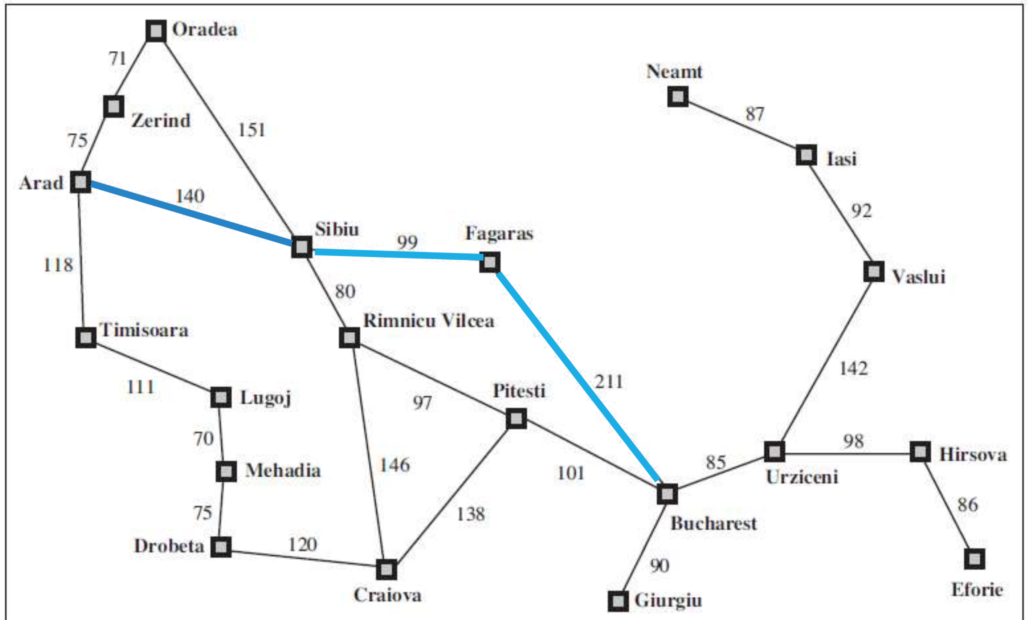


Figure 3.2 A simplified road map of part of Romania.

BFS goal path

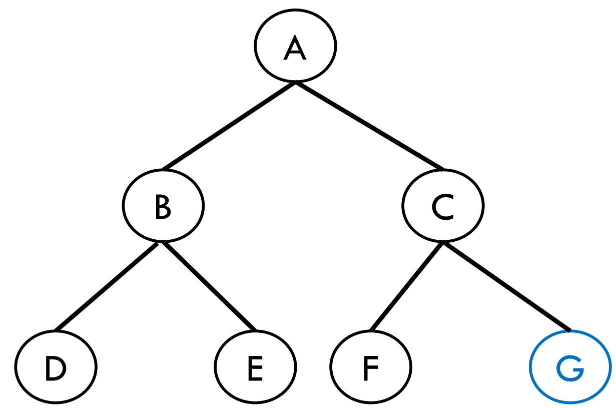
BFS: TIME COMPLEXITY?

Worst complexity is when G is a goal state.

In this case, total number of nodes generated is

$b+b^2+b^3+...+b^d = O(b^d)$ (The d^{th} layer contains nodes much larger than all the nodes in previous layers combined!)

So the time complexity is $O(b^d)$



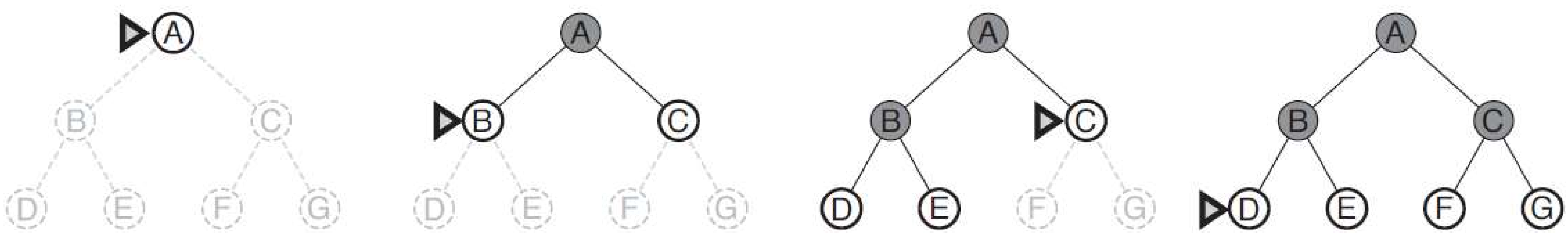
A binary tree, b=2, d=2

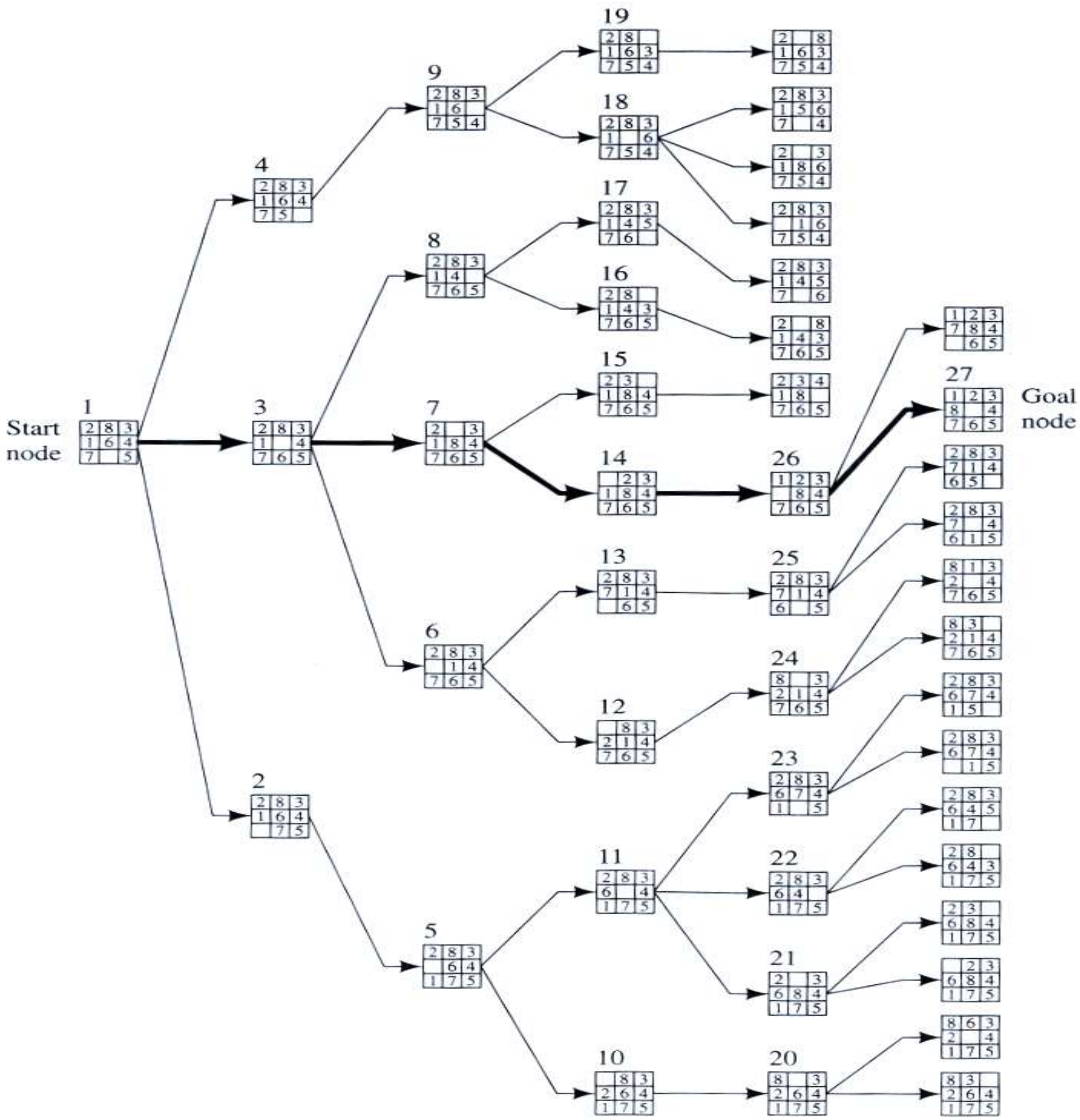
BFS: SPACE COMPLEXITY?

There will be $O(b^{d-1})$ nodes in the explored set and $O(b^d)$ nodes in the frontier.

So the space complexity is $O(b^d)$, i.e., it is dominated by the size of the frontier.

Exponential time complexity can be accepted but exponential space complexity is BAD !





Start node
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
Goal node