Advanced
Artificial Intelligence

Ethics assignment: Presentation & Report

- You are all in groups of 4 students
- Each group has a topic assigned

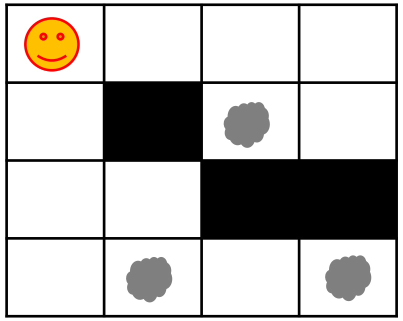
- Presentation 21st October 08:15-12:00 (10 min per group)
- Deadline for report 26th October 08:00 (5 pages per group)

- More instruction can be found on the CANVAS page.

- Be aware of the university's plagiarism rules

Intelligent agents

Autonomous
Vacuum
cleaner

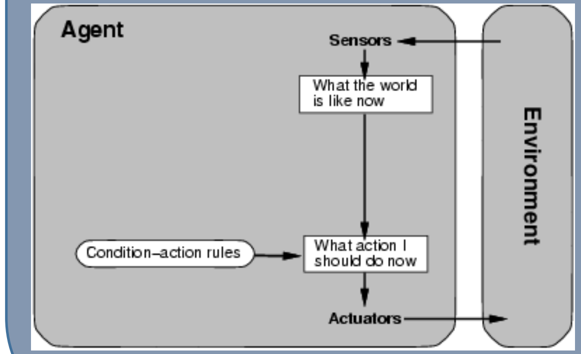


Simple reflex agent

If dirt->suck

If bump -> turn right

If not dirt and not bump -> move forward



Model-based reflex agent

Can build up a map of the world

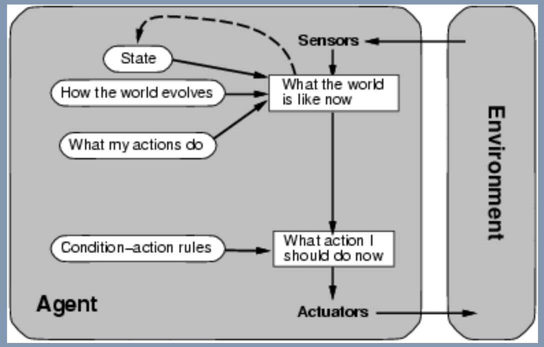
Can learn the layout of your home

Can learn where there are hinder

If bump mark that space in map as ”hinder”

If dirt-> suck and mark space in map as clean

…



Goal-based reflex agent

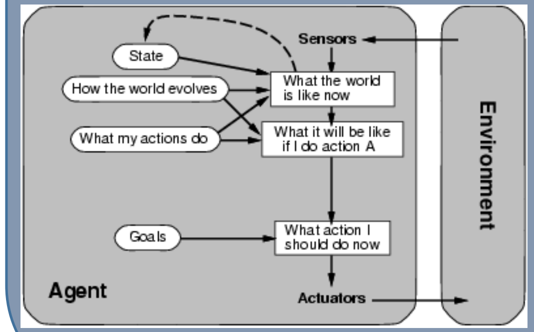
Can build map of the world, etc.

Goal to clean the whole apartment

Counts: how many unvisited fields are there?

Goal: no unvisited fields

Every time it does an action it choses the action that brings it closer to the goal (i.e. an action the decreases the unvisited field counter)



Search

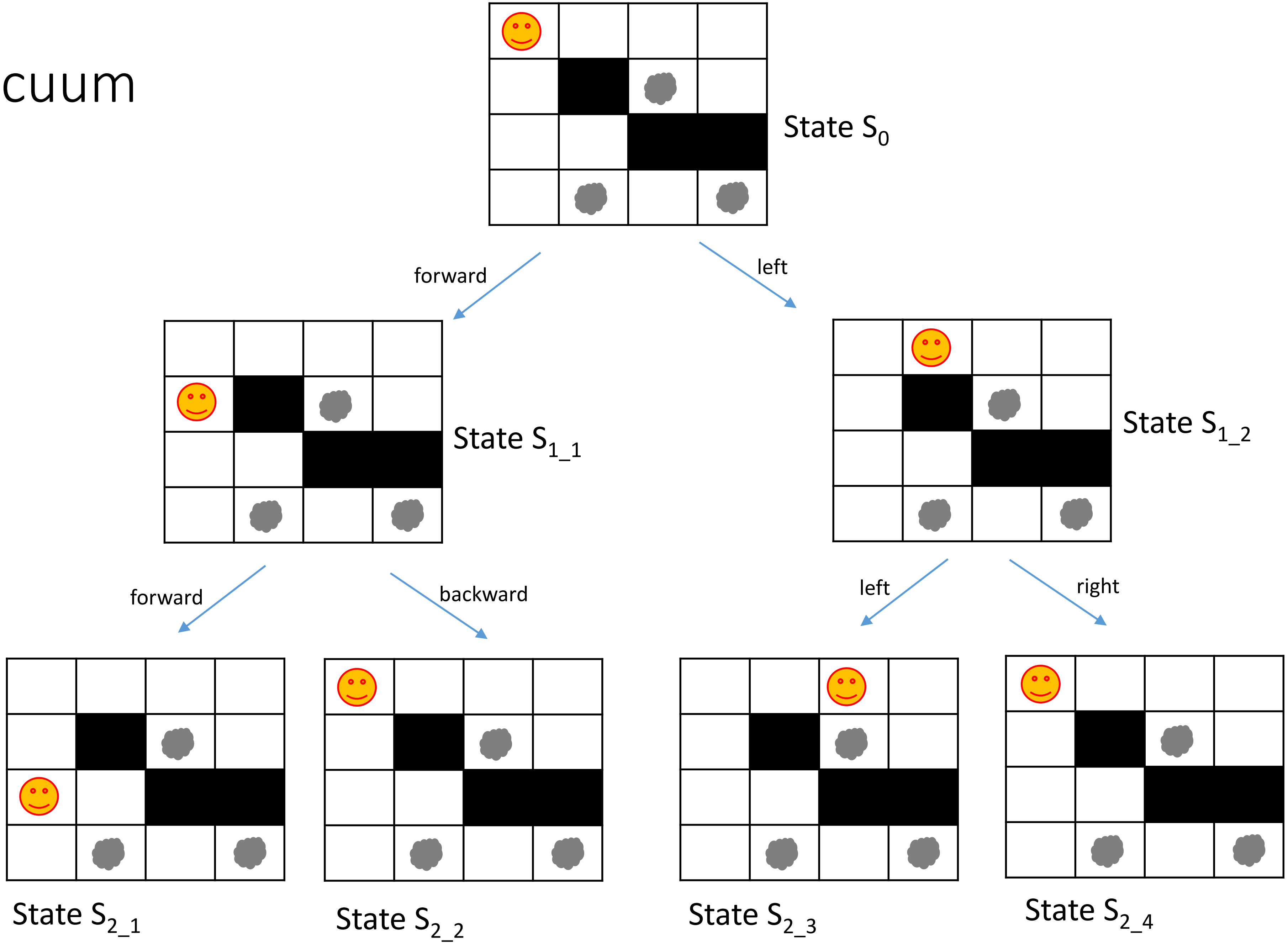
How to find the best way from an initial state to a goal state?

Autonomous vacuum cleaner

- Available actions
  - Go left
  - Go right
  - Go forward
  - Go backward

- Applicable actions
  - Go forward
  - Go left (seen from the agent's perspective)

- How does it know the applicable actions in the given situation?



1997 IBMs Deep Blue beats human chess master

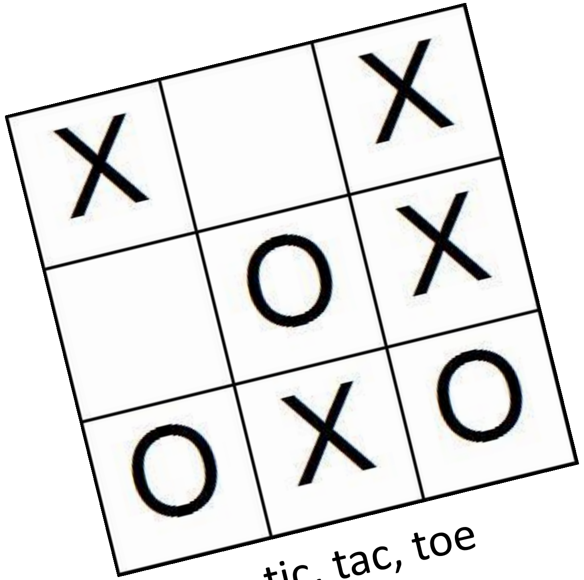
https://www.youtube.com/watch?v=NJarxpYyoFI

Garry Kasparov, chess master

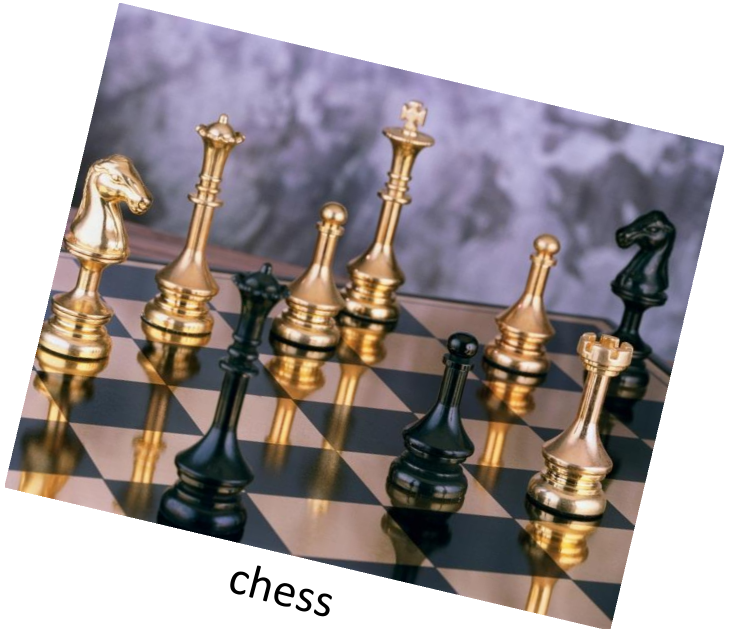
Deep Blue, at the Computer History Museum

Brute Force

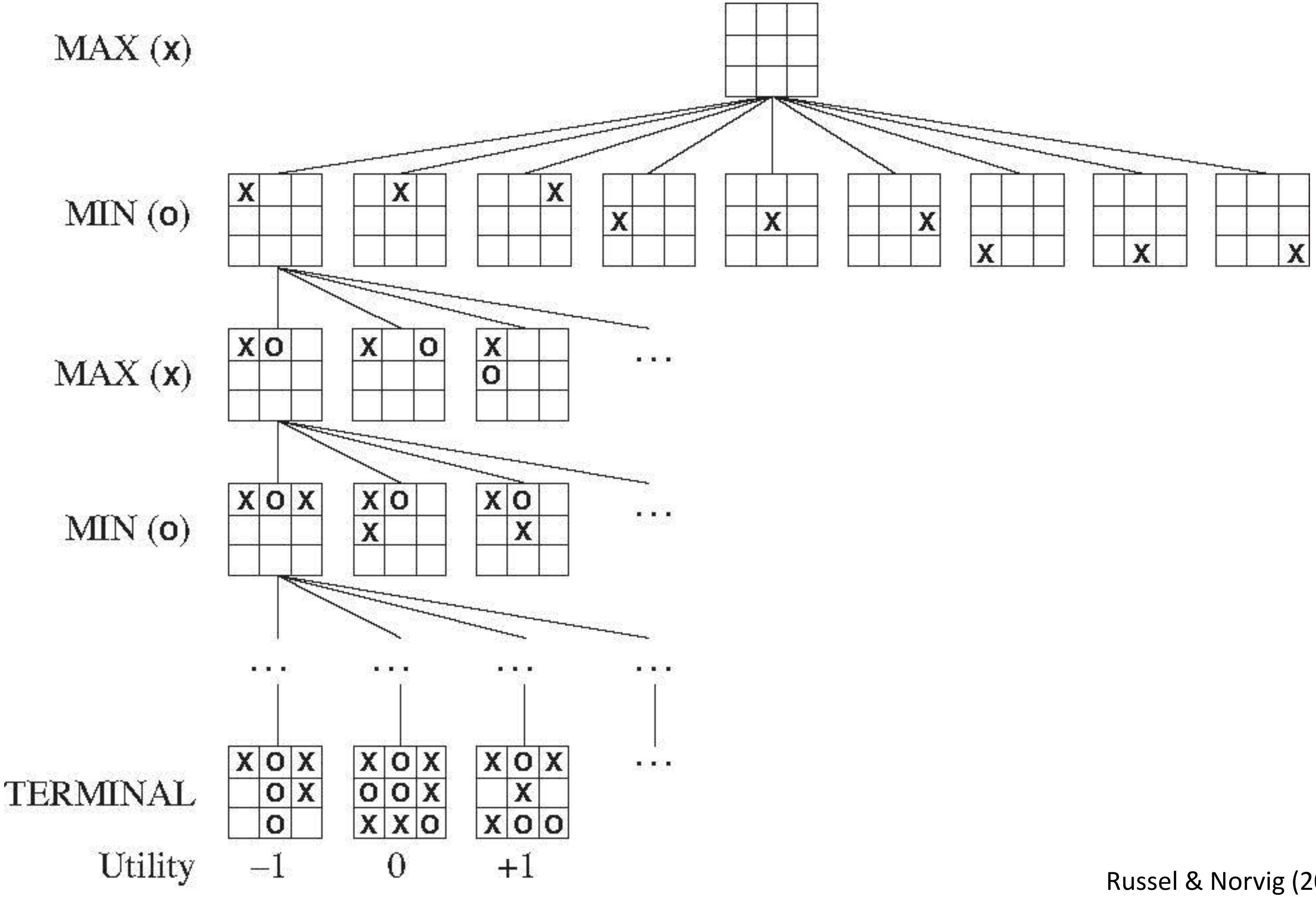
- Looking at every possible alternative
- Deciding which one is the best
- Doing it



tic, tac, toe

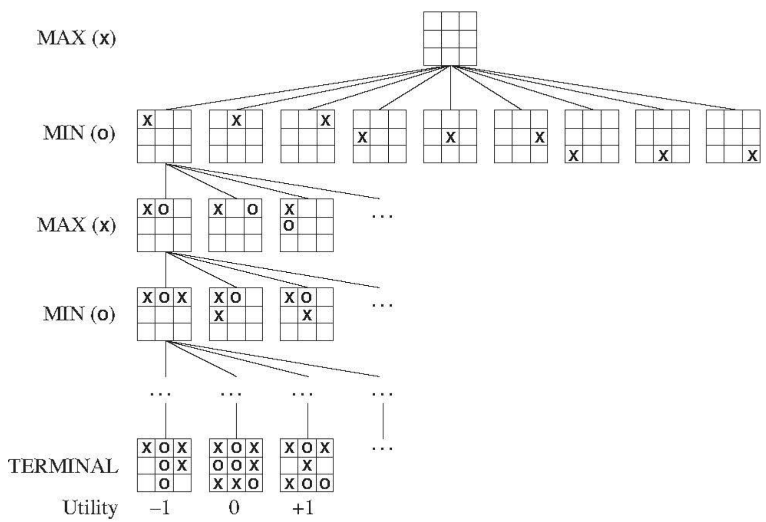

chess

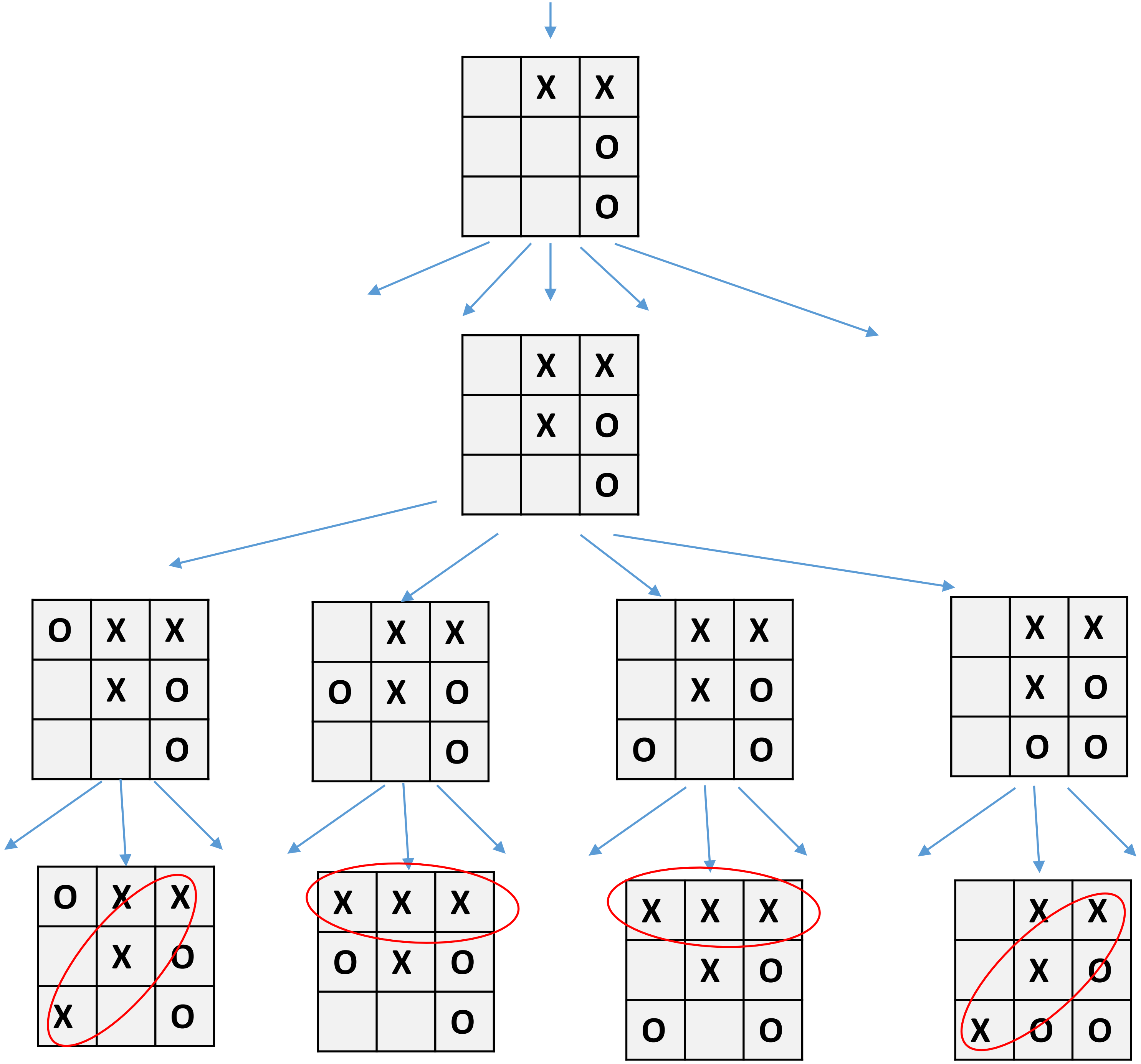
Searching for the best move:



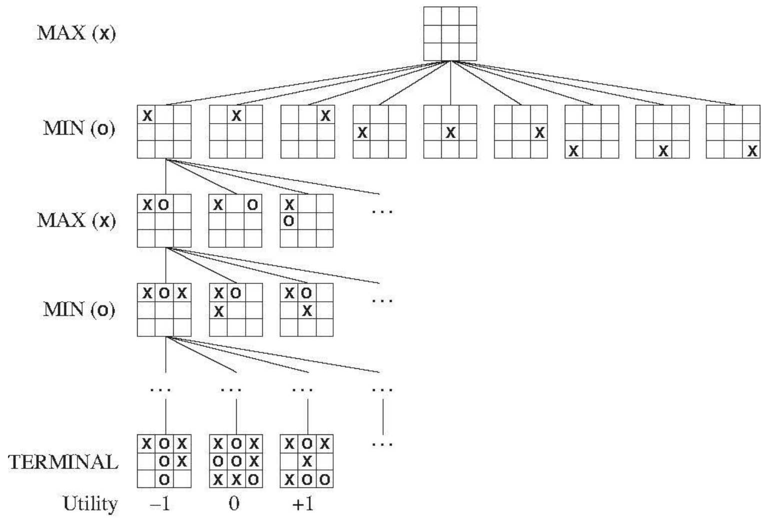
Russel & Norvig (2010)

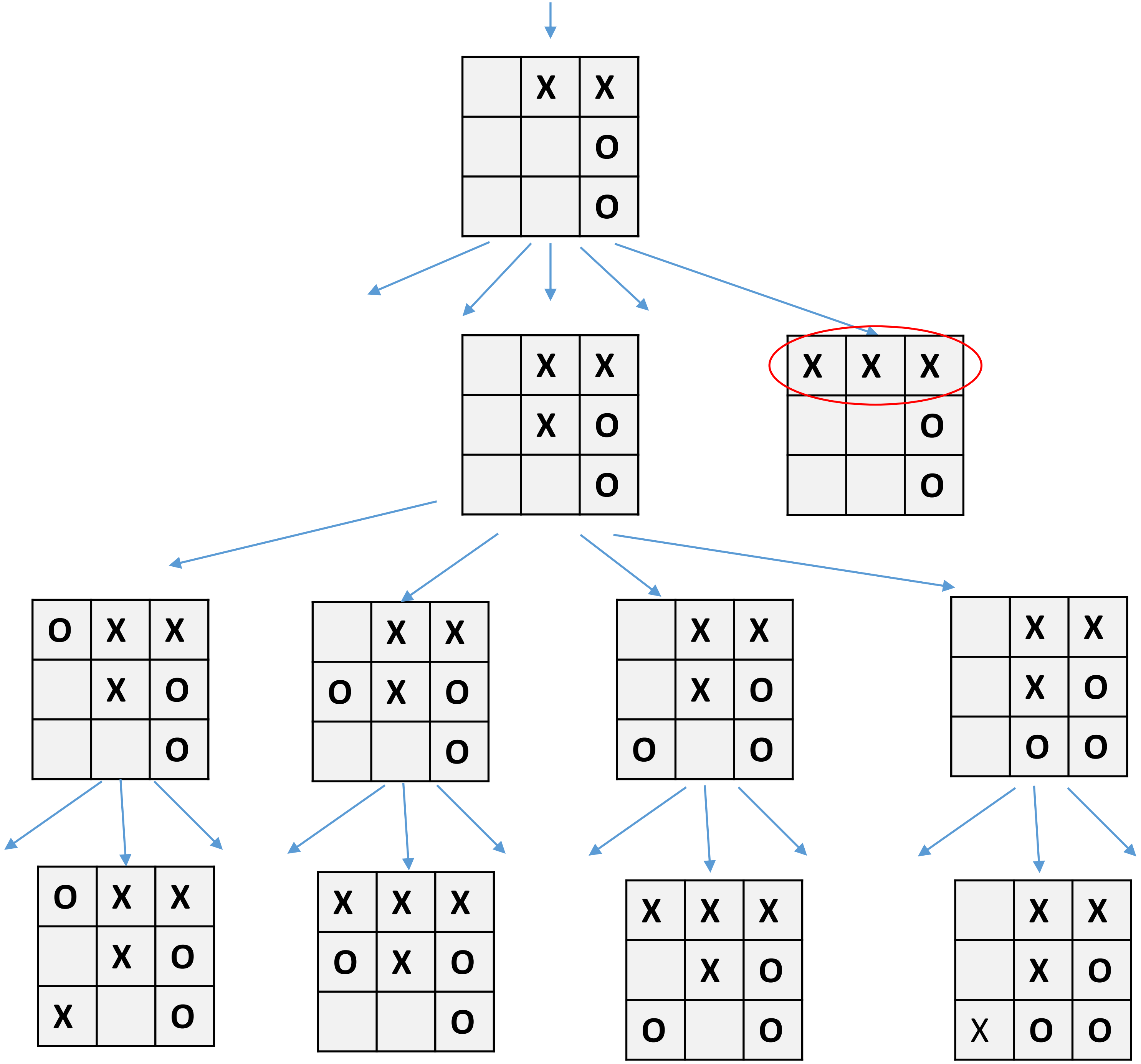
Searching for the best move:





Searching for the best move:





Search in AI

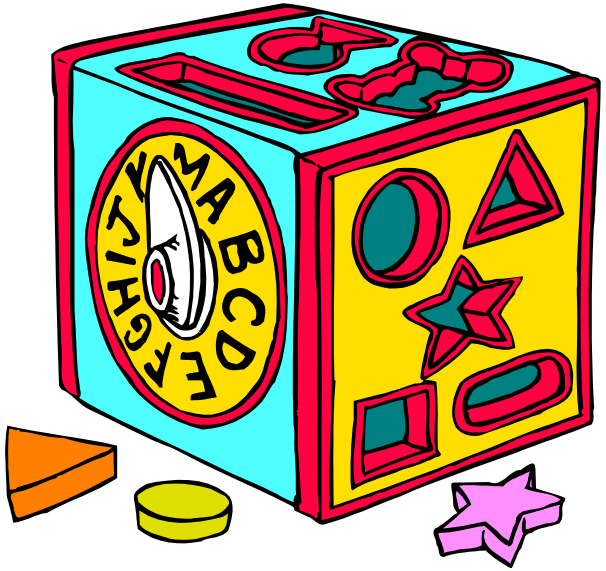
Finding the right action to perform given what is known about the situation.

Finding the best next move (e.g. Chess).

Develop the best plan (e.g. packing furniture's in the van so that they fit)

Find the best route (GPS)

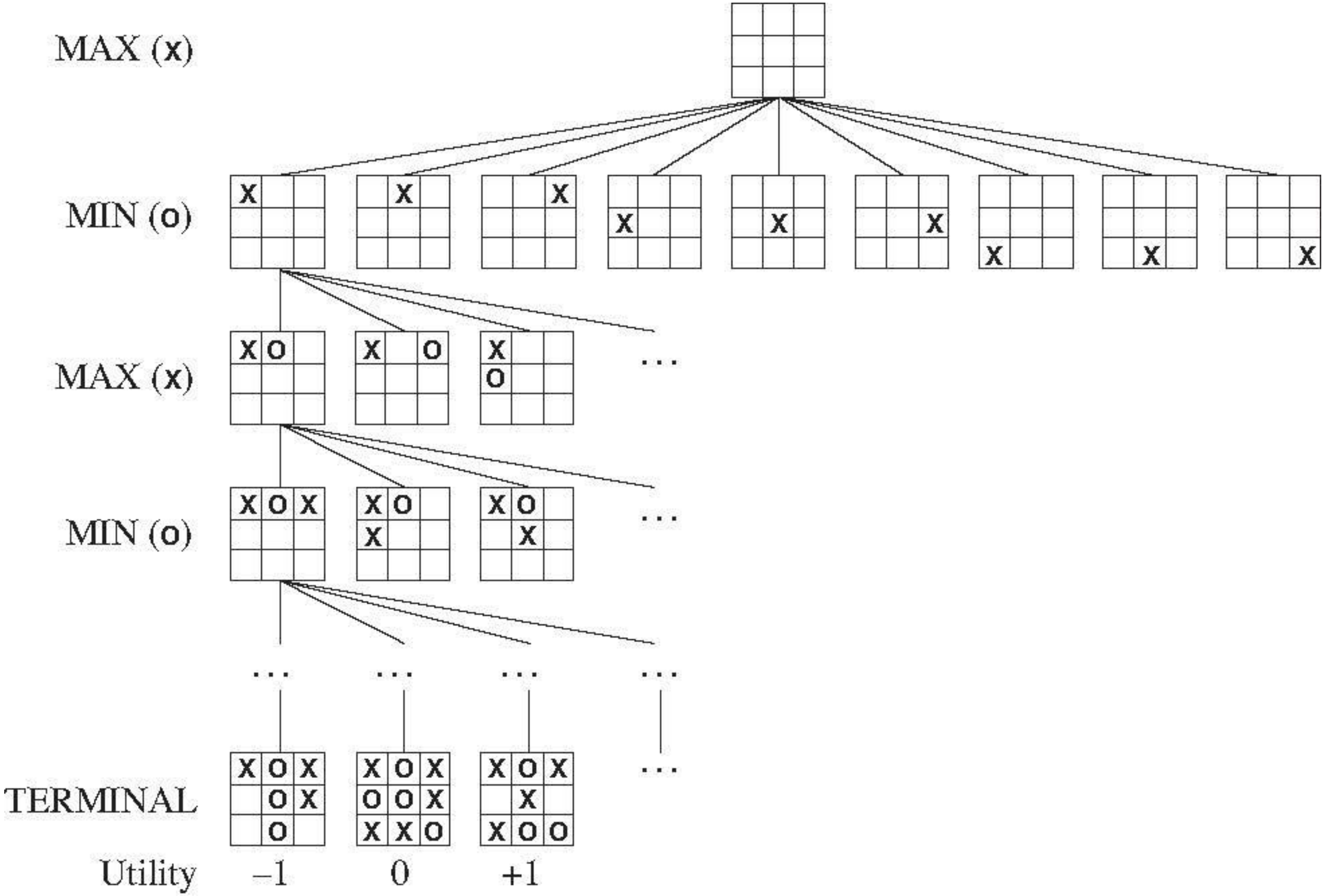
Examples of search problems



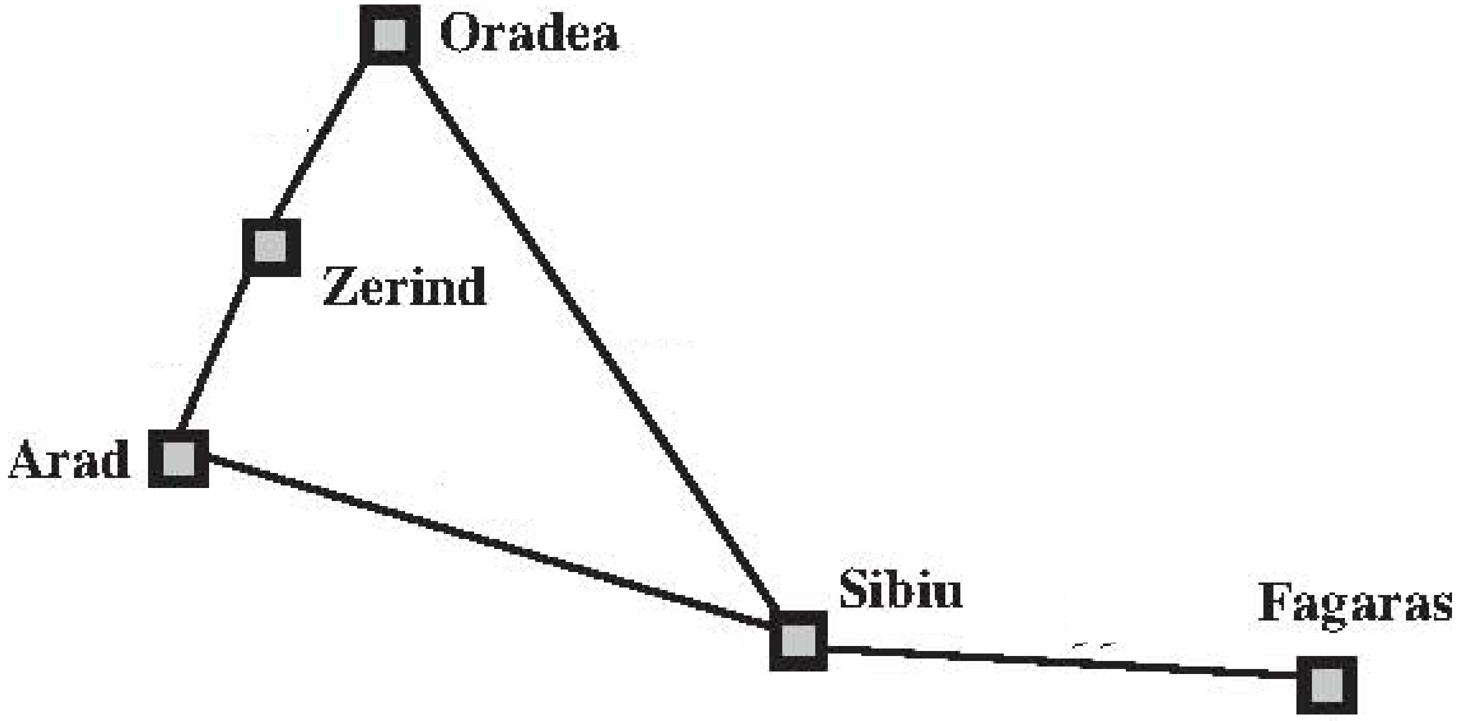
Search in AI

1. Create the search space (search tree)
2. Decide on the best method to search the tree

Searching for the best move in tic, tac, toe:

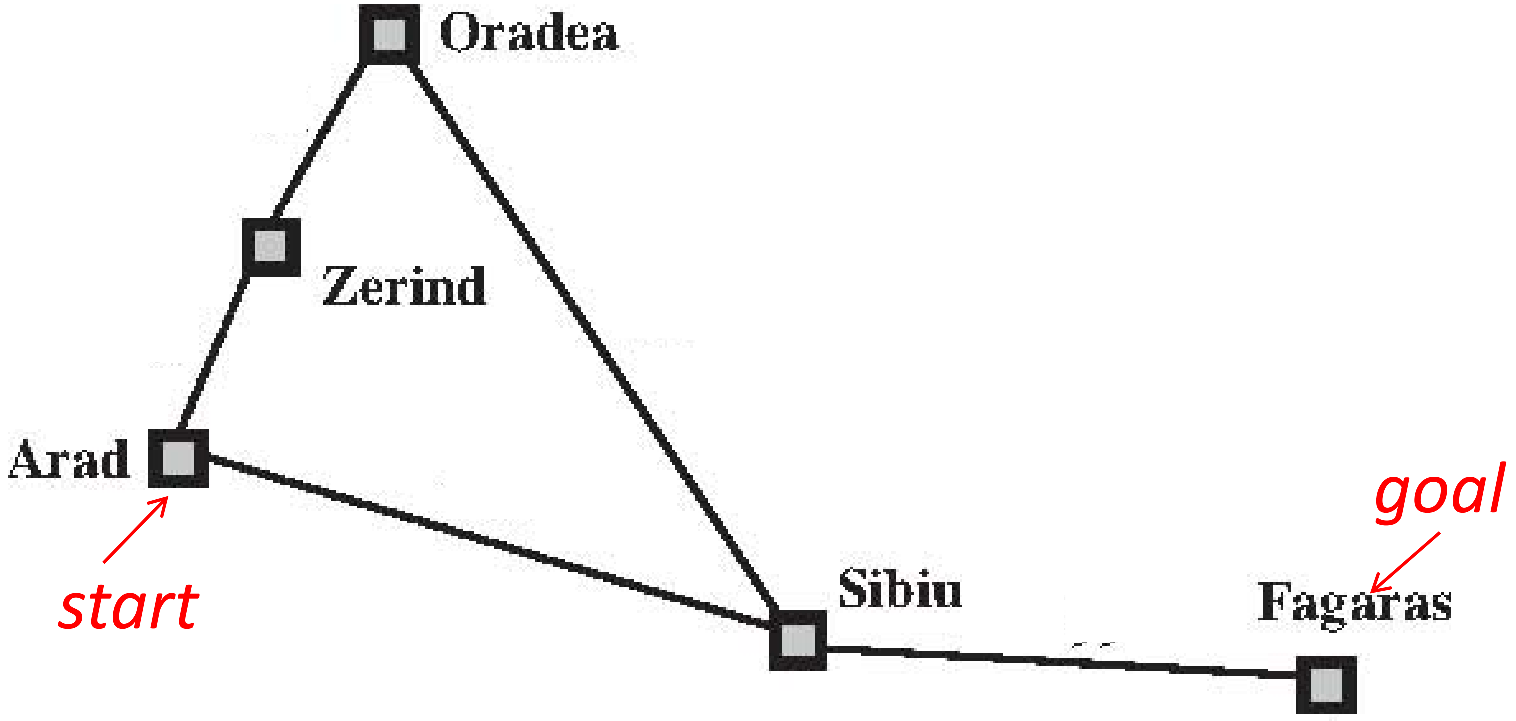


Route planning



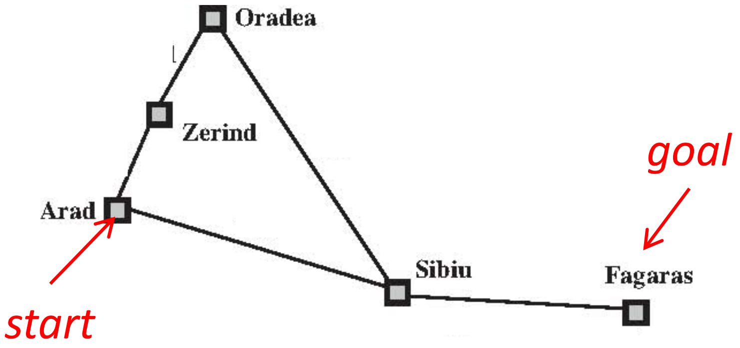
Part of abstract map of Romania (Russel, Norvig)

Route planning

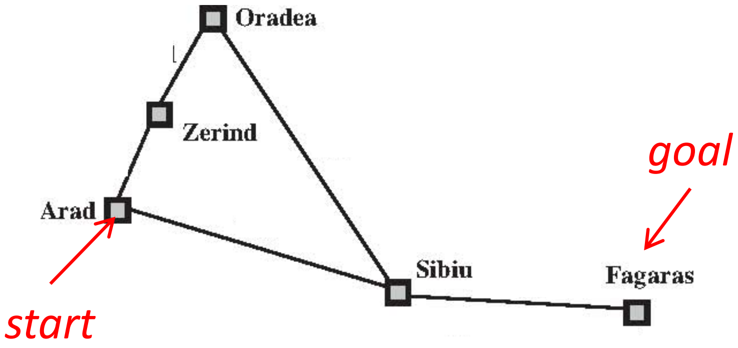


Part of abstract map of Romania (Russel, Norvig)

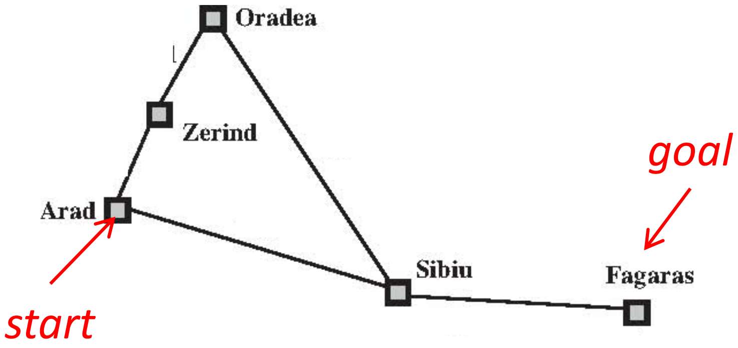
Building the search space



Building the search space



Arad

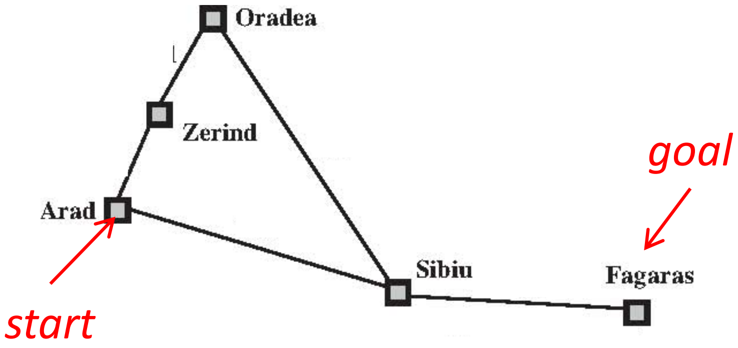


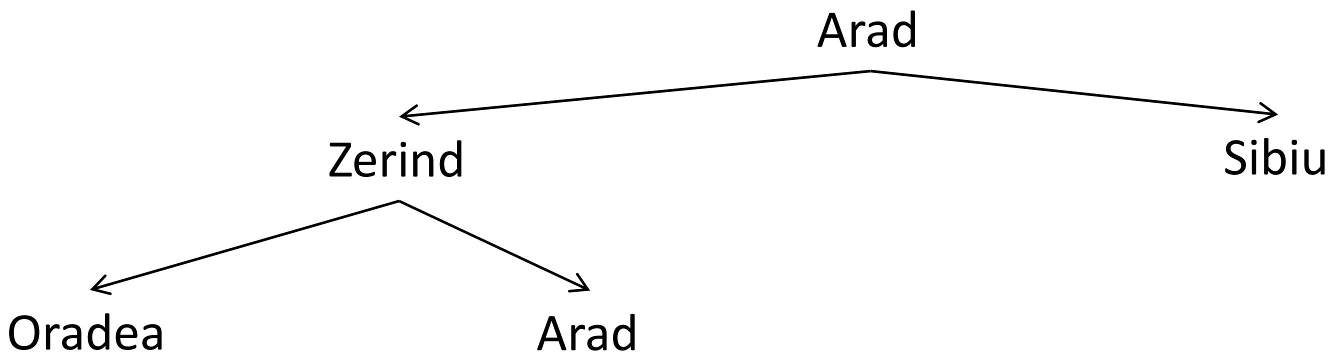
Building the search space

Arad

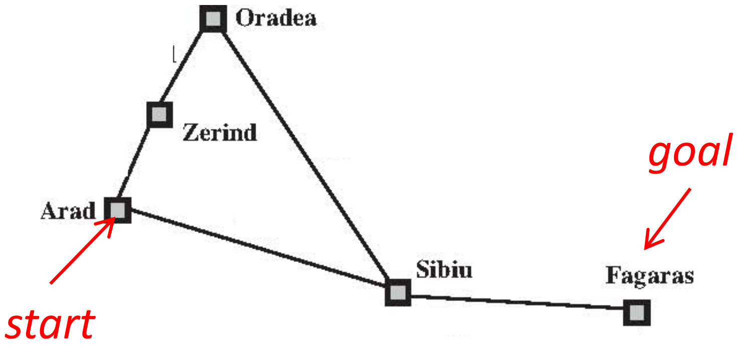
Zerind Sibiu

Building the search space



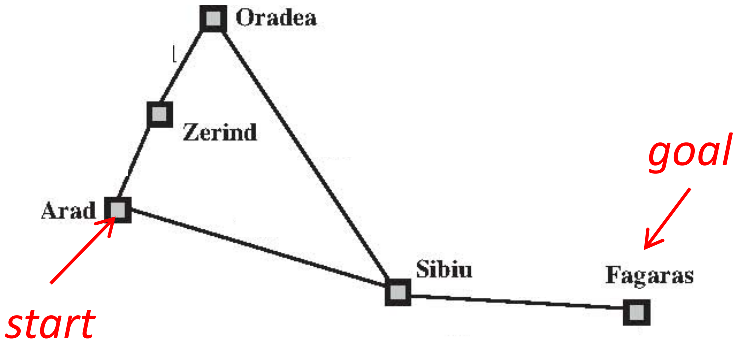


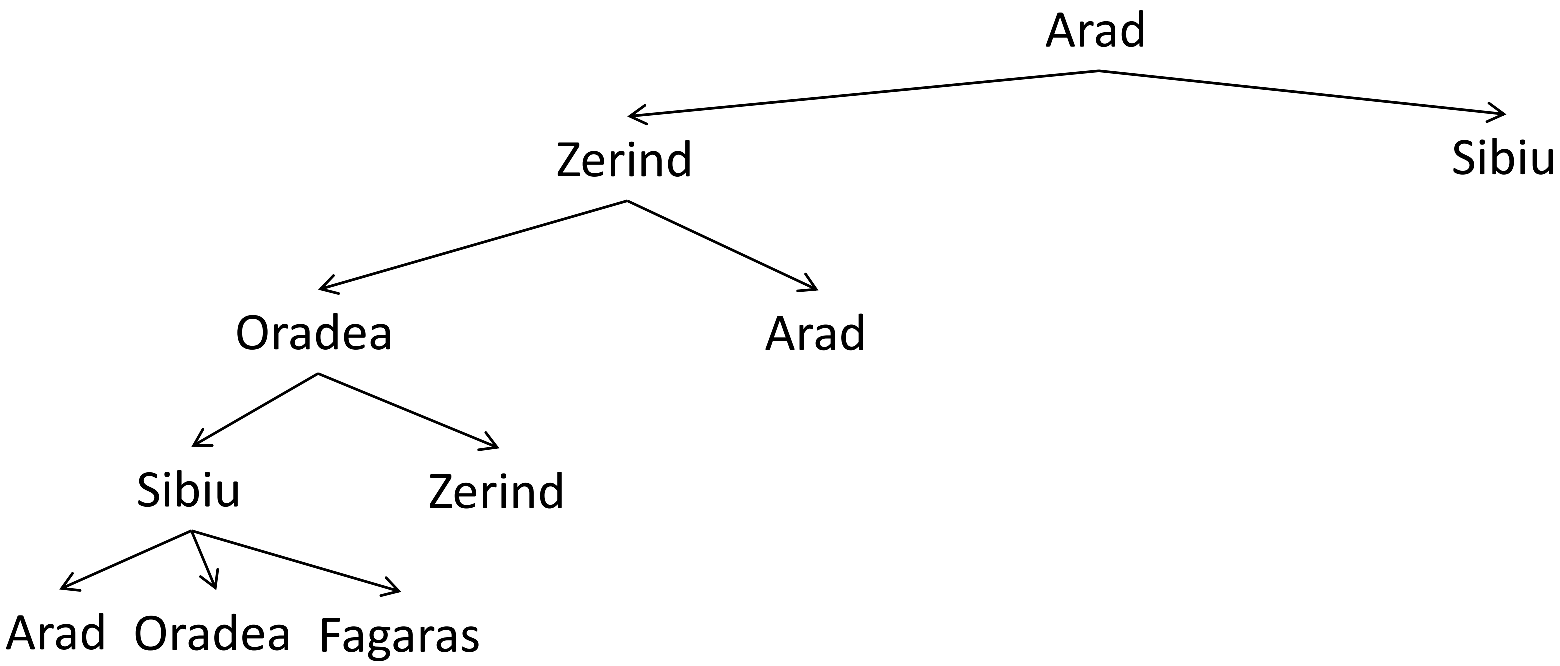
Building the search space



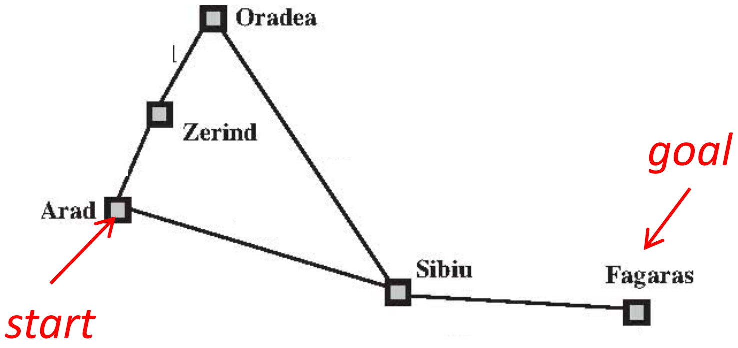
- Arad
  - Zerind
    - Oradea
      - Sibiu
      - Zerind
    - Arad
  - Sibiu

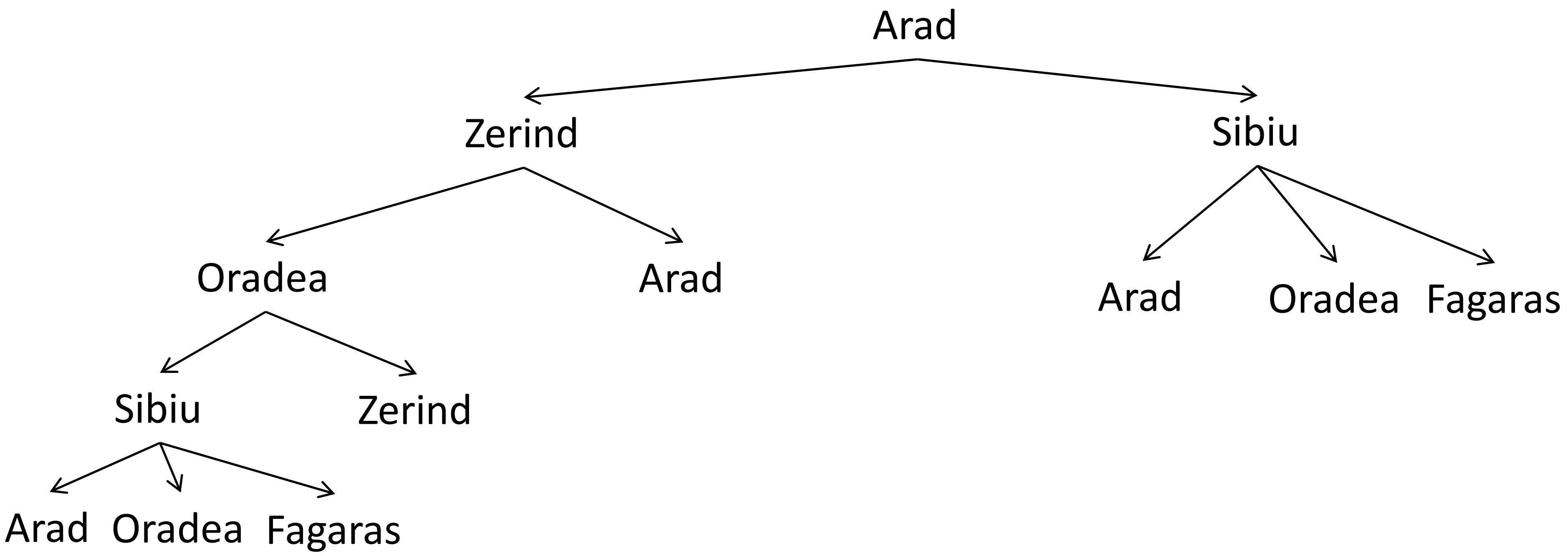
Building the search space



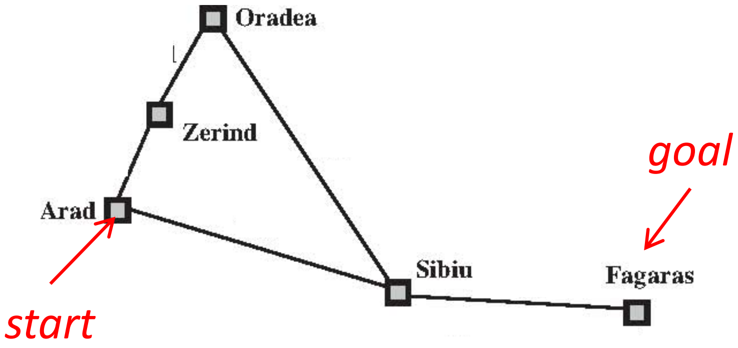


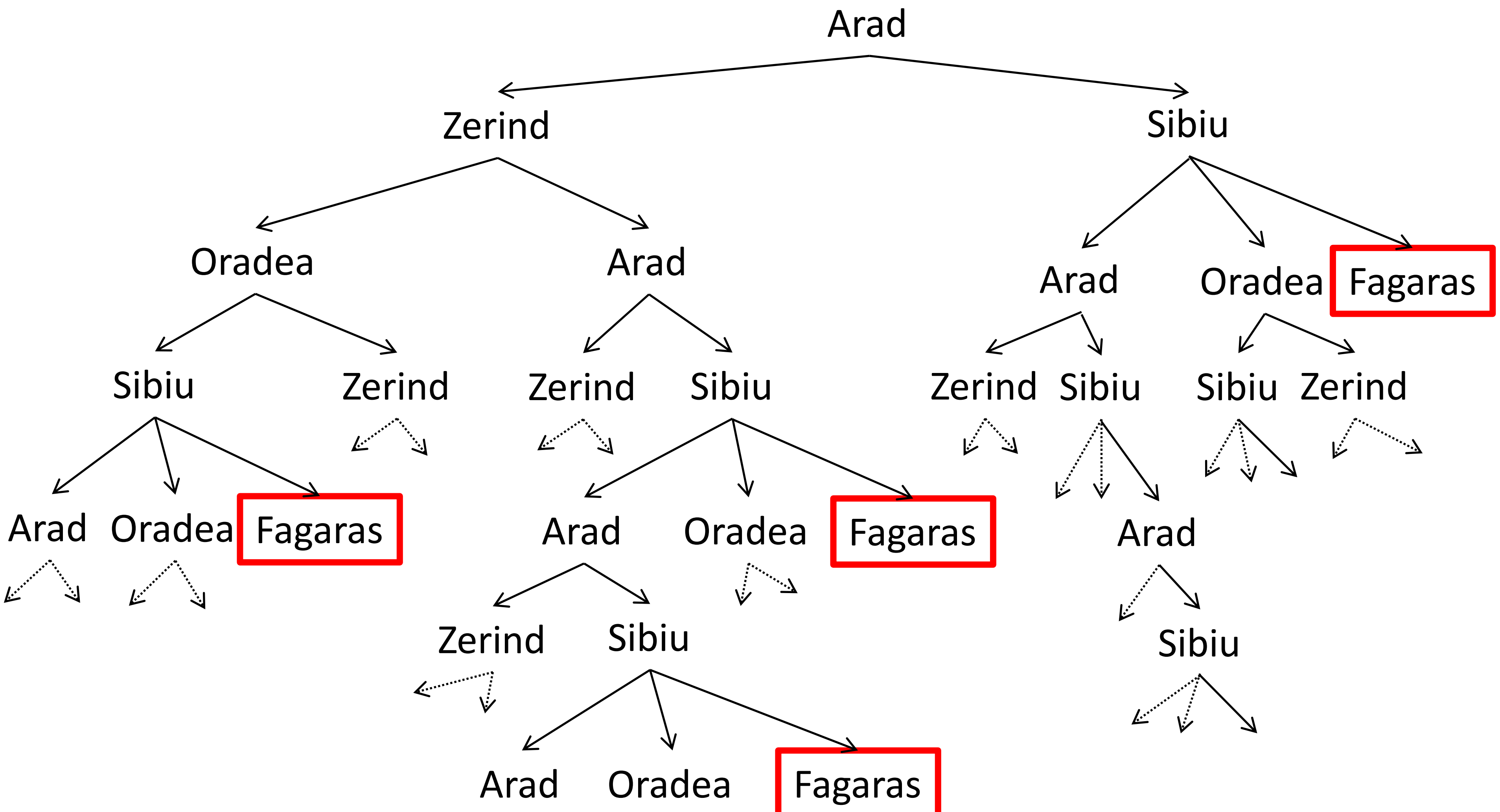
Building the search space



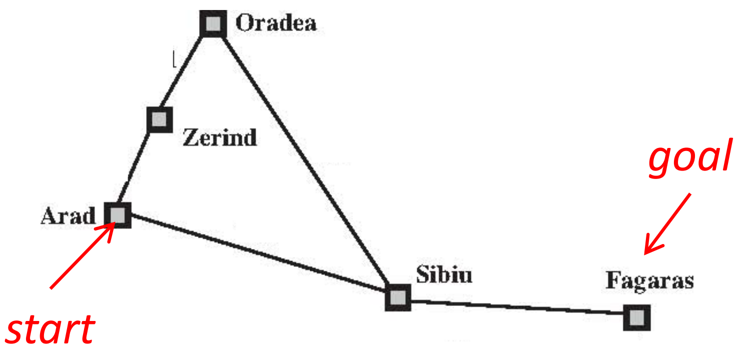


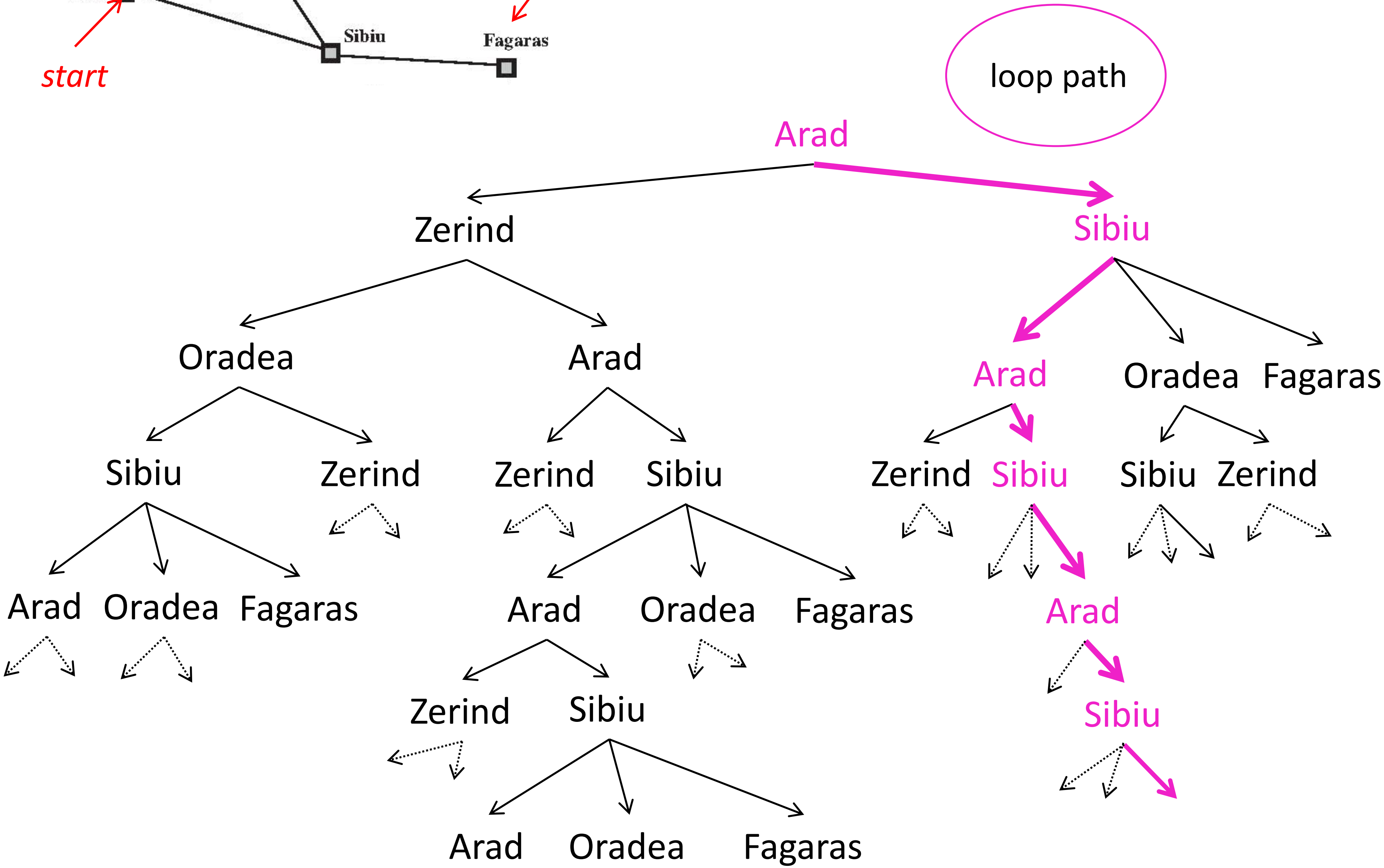
Building the search space



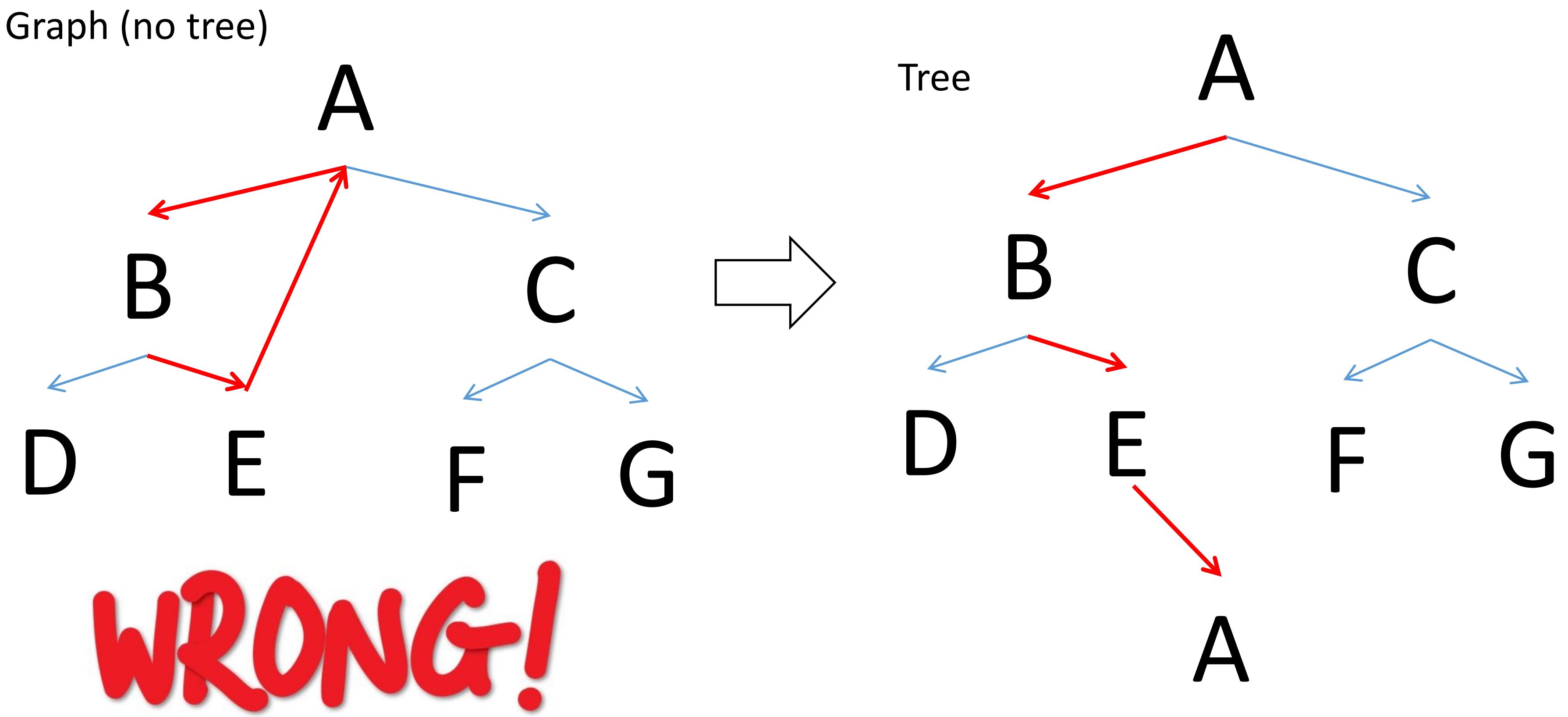


Building the search space

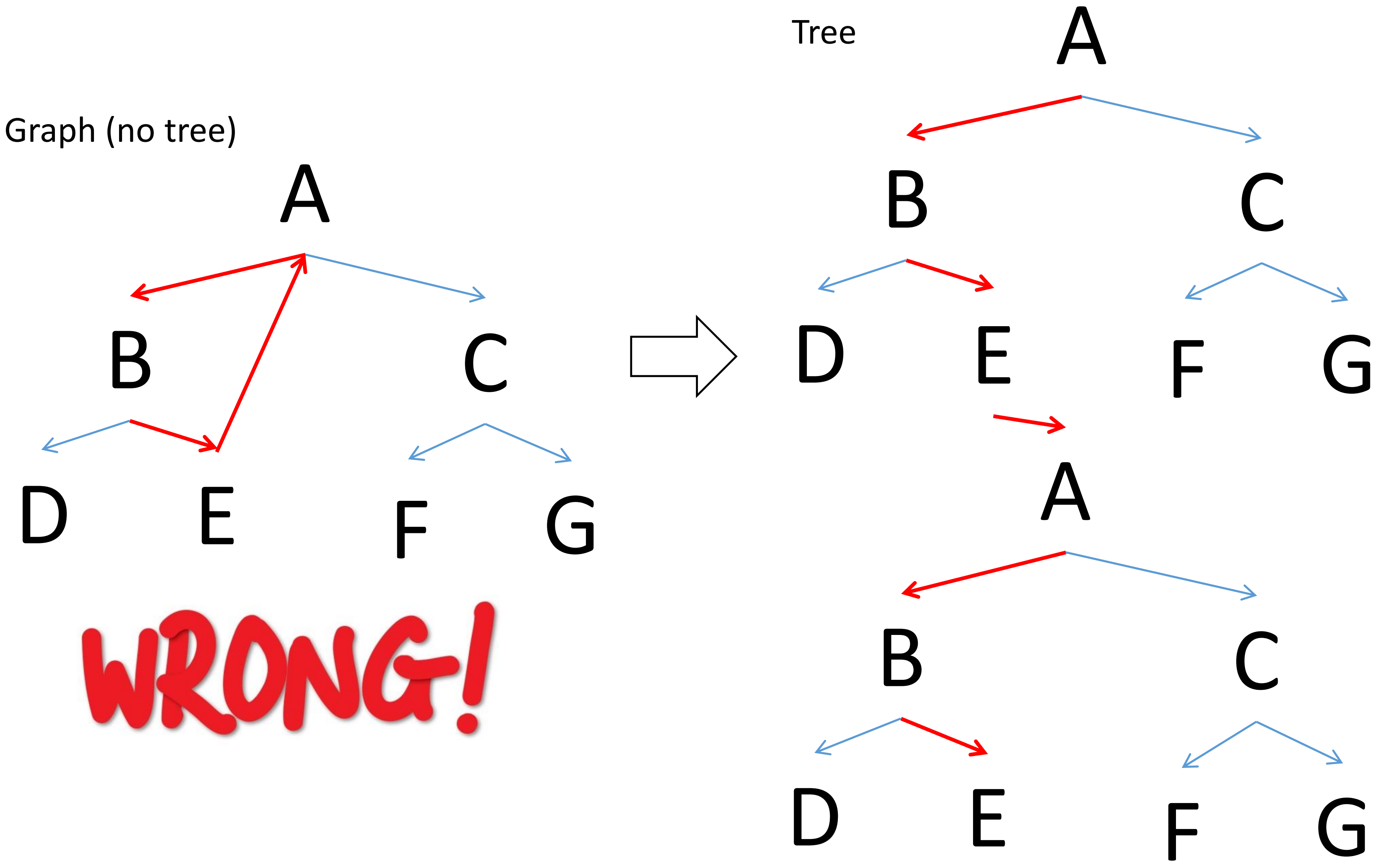




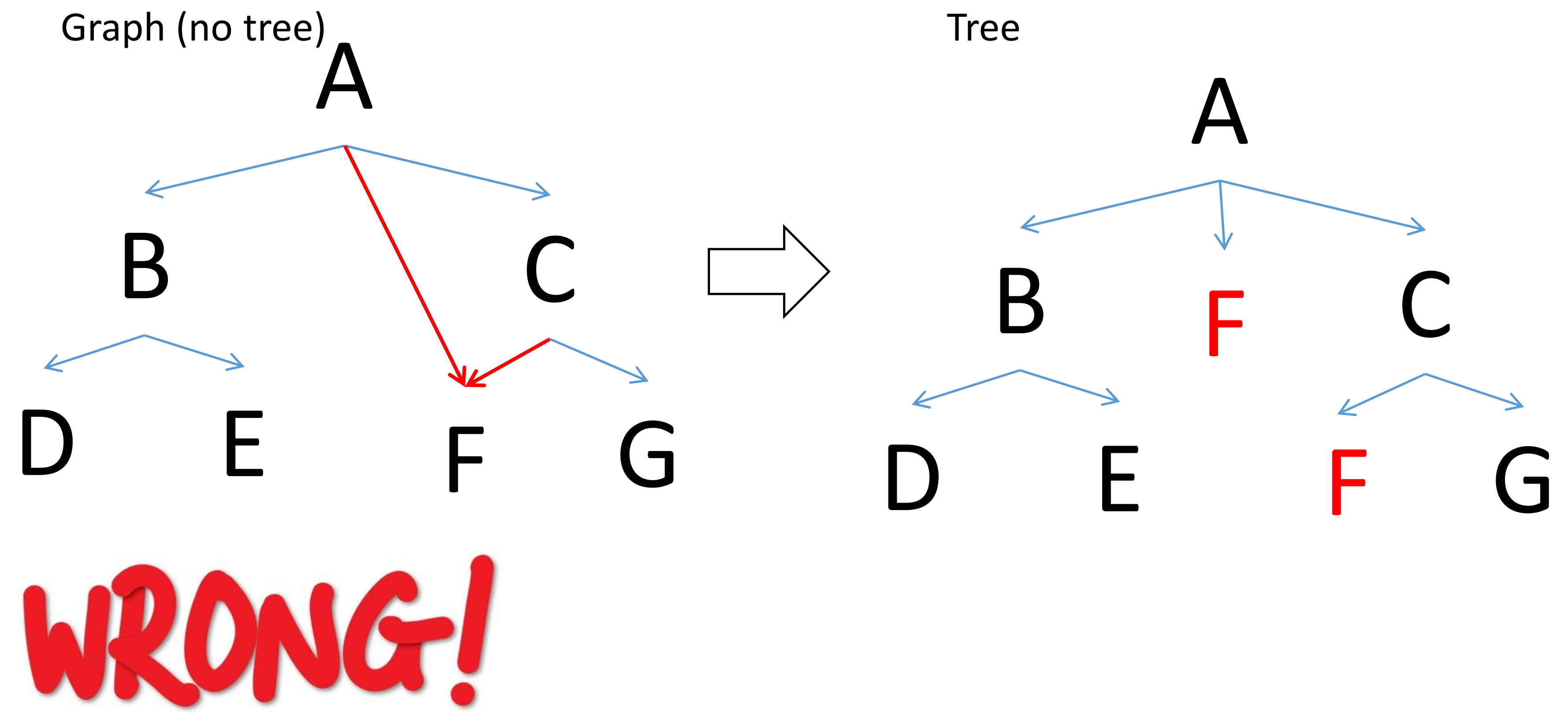
A tree is a graph without loops



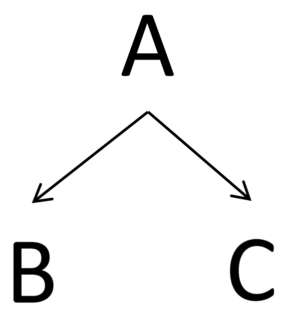
A tree is a graph without loops



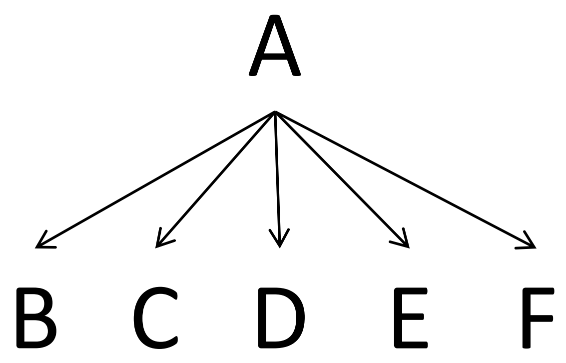
A tree is a graph without loops



Braching factor



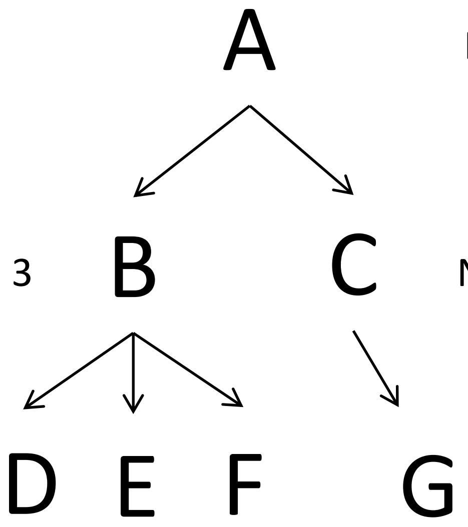
Braching factor = 2



Braching factor = 5

Nonuniform braching factor

Node A has braching factor 2

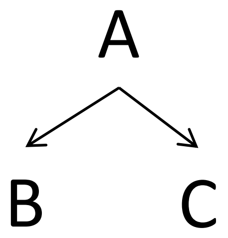


Node B has braching factor 3

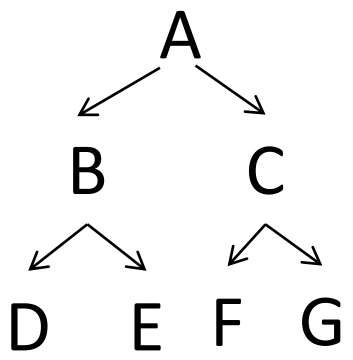
Node C has braching factor 1

The tree has nonuniform braching factor of 1- 3
For calculation use the *average braching factor* (in this case 2)
or the *worst case braching factor* (in this case 3)

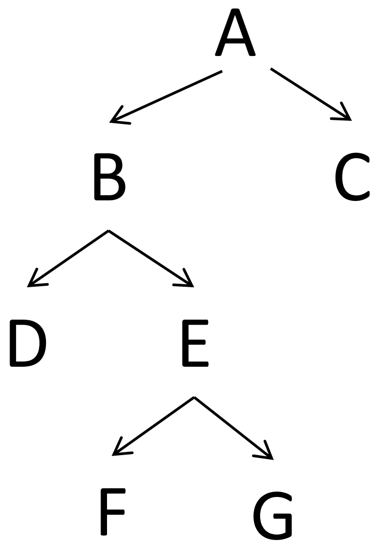
Depth



Depth = 1



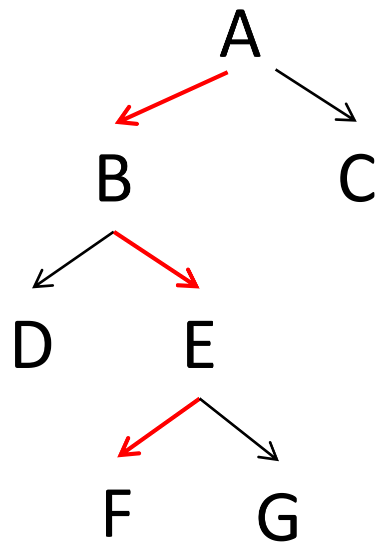
Depth = 2



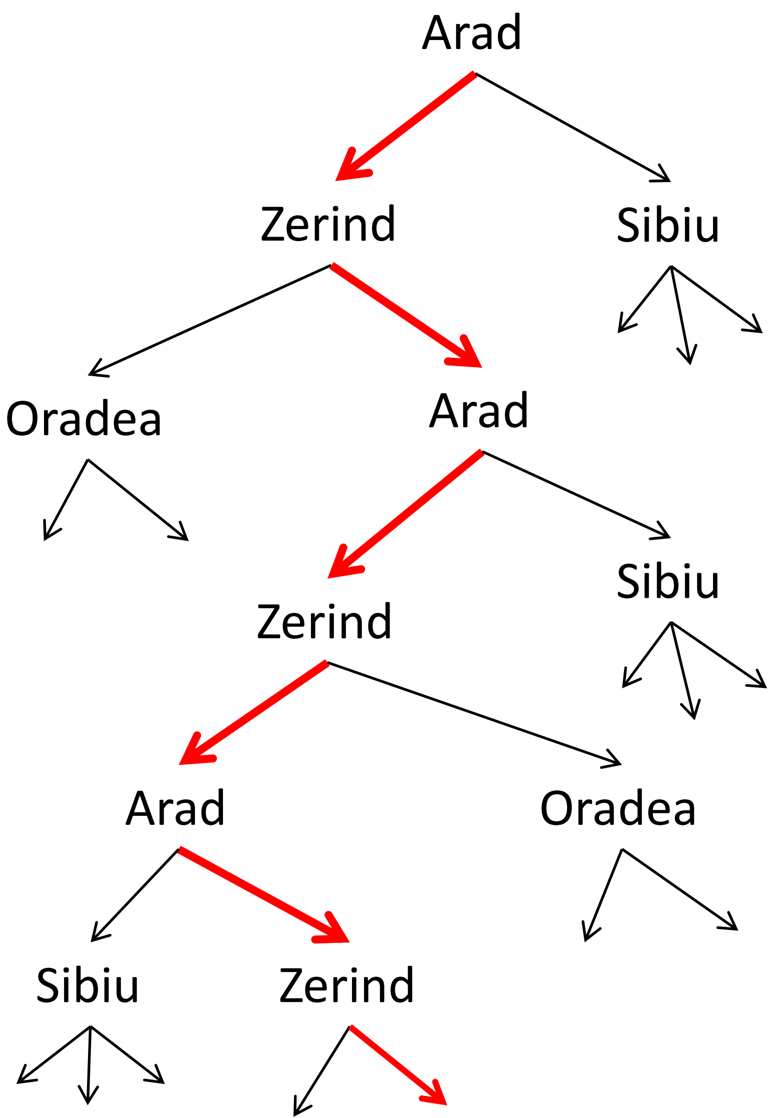
Depth = 3

Path

Path from A to F



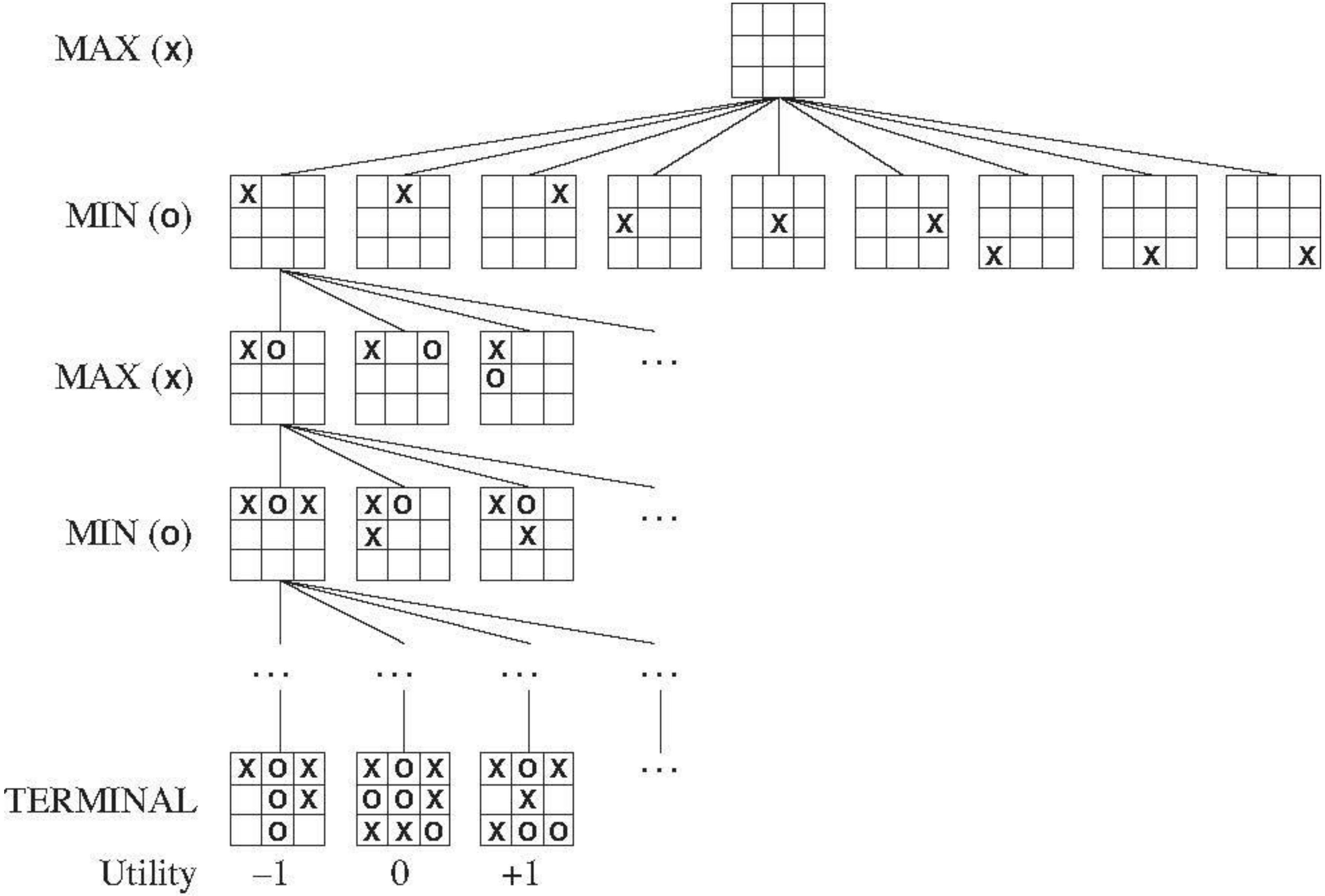
Loop path


Arad
Zerind
Sibiu
Oradea
Arad
Zerind
Sibiu
Arad
Oradea
Sibiu
Zerind

Searching for the right solution

- Try “actions” in any combination until you find the “solution” to the problem.
- Use some strategy to go through all possible action combinations.
- If possible, use a strategy to minimize the action combinations you have to try.

Searching for the best move in tic, tac, toe:

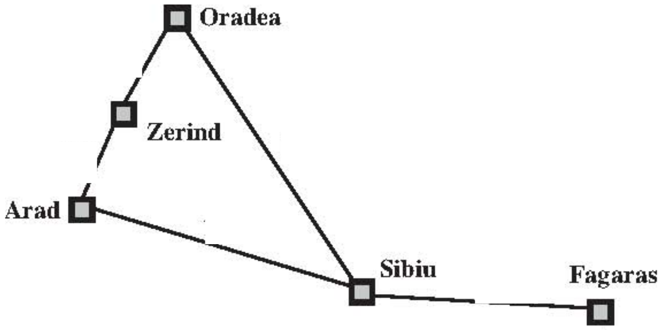


Uninformed Search

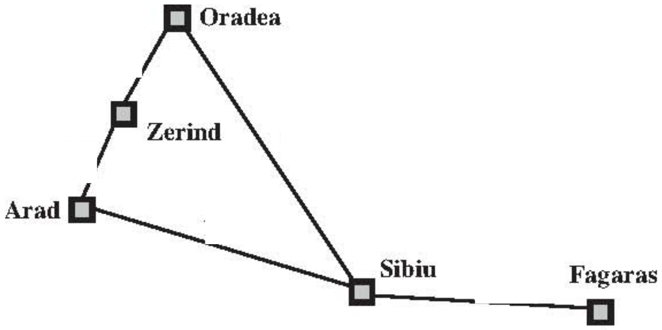
- Only the search tree is present
- There is no knowledge what the best path might be
- Build the search space (tree) as you go from your representation of the problem (e.g. hash table)

- Breadth first search
- Best first search
- Depth first search
- Depth limited search
- Iterative deepening

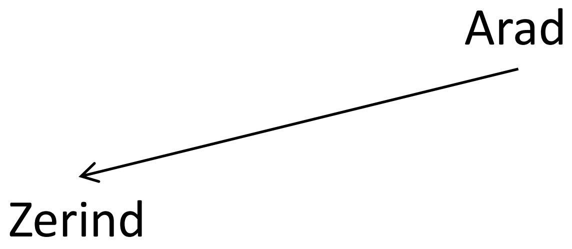
Breadth-first search

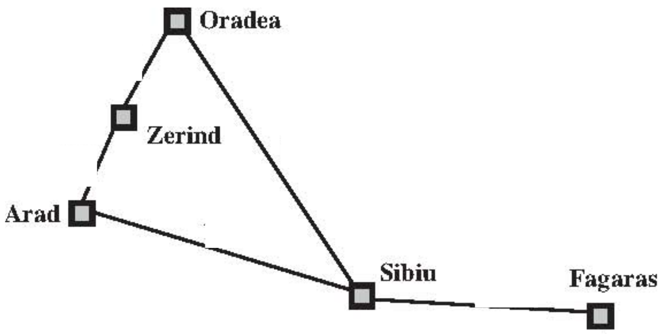


Arad



Breadth-first search





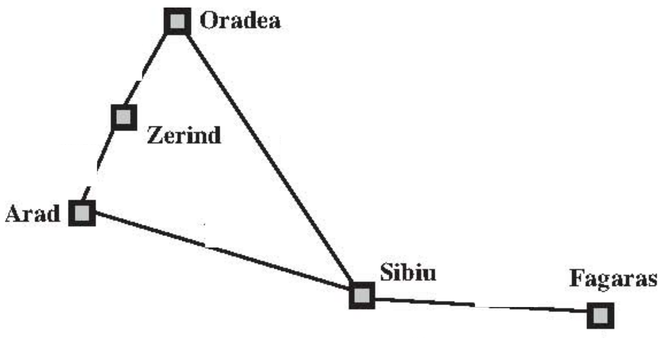
Breadth-first search

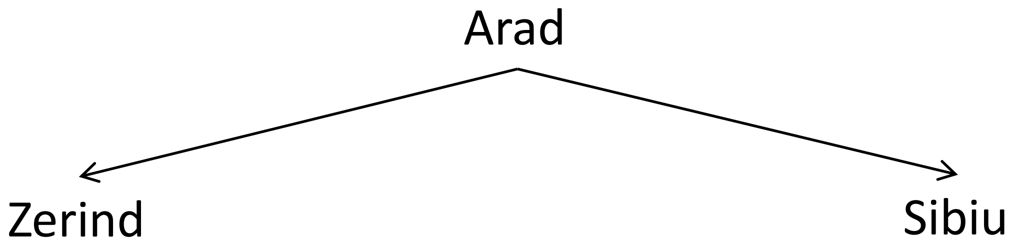
Arad

Goal?

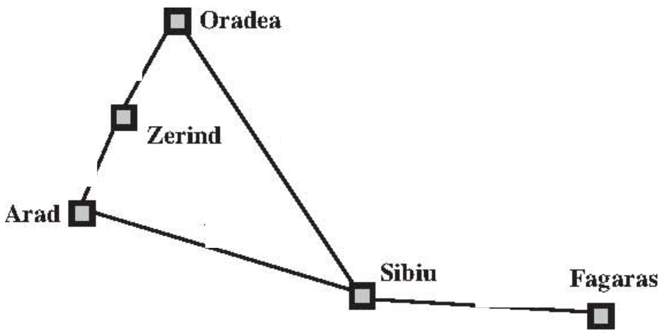
Zerind

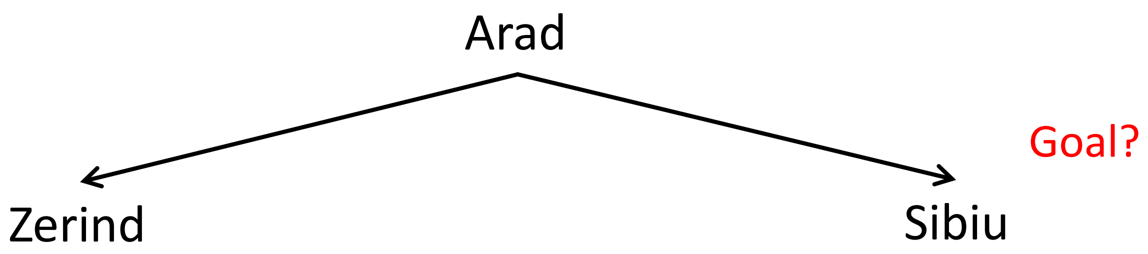
Breadth-first search



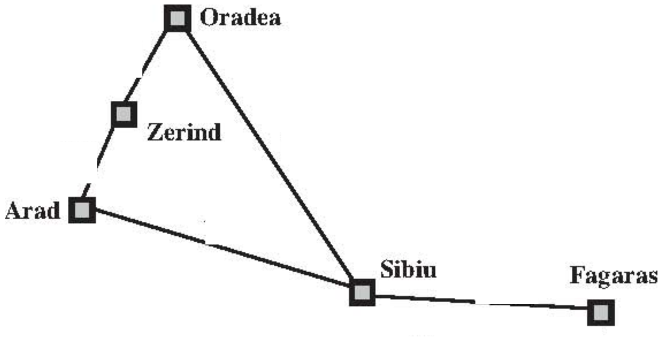


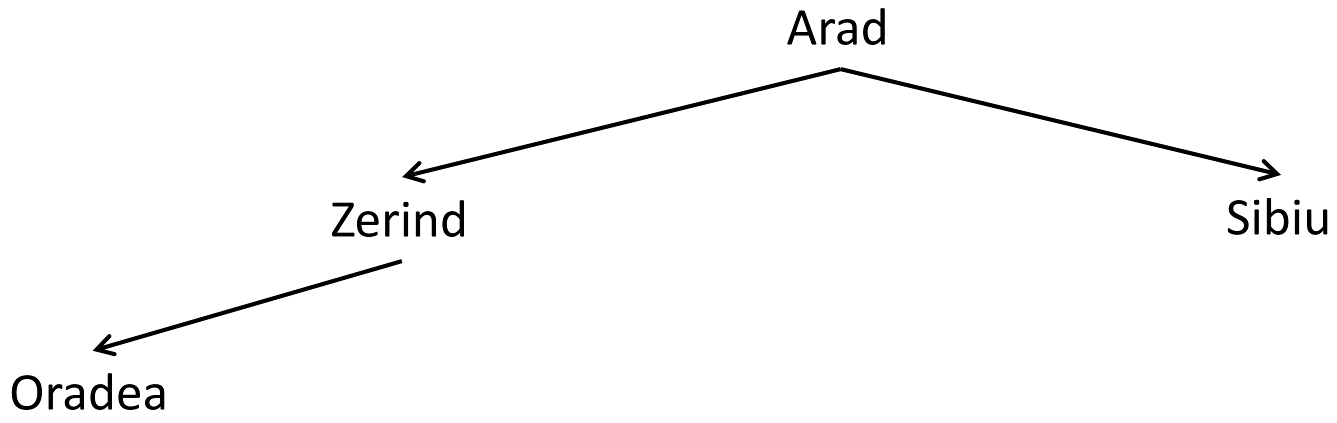
Breadth-first search



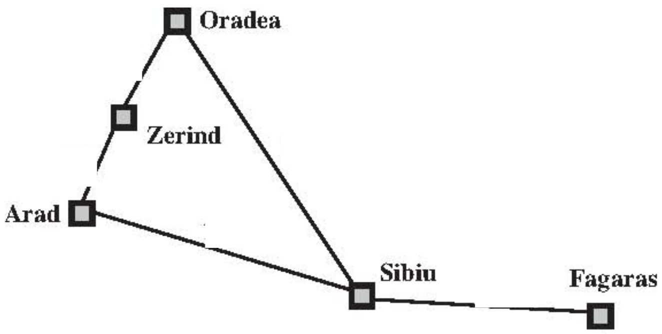


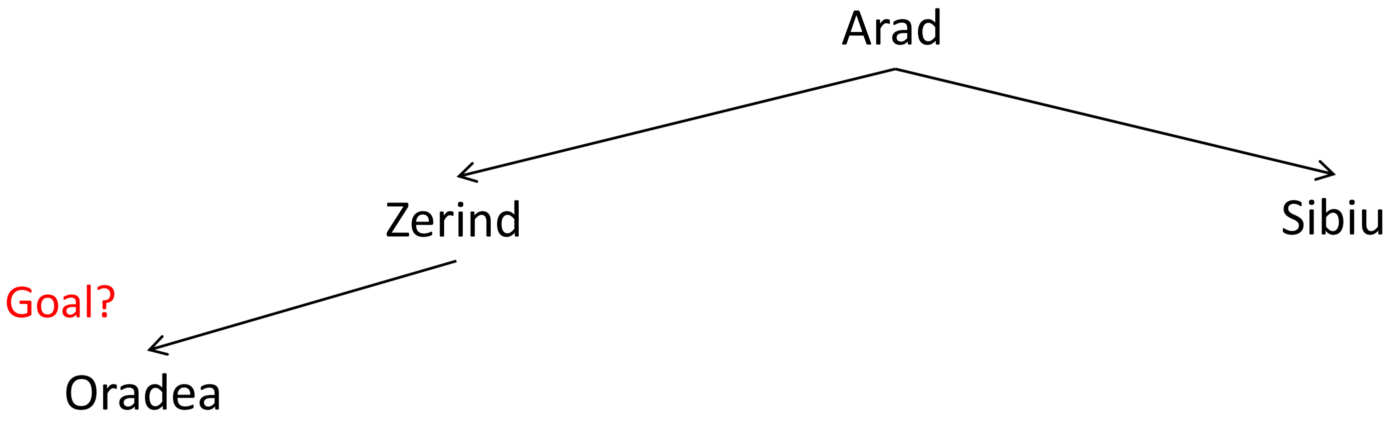
Breadth-first search



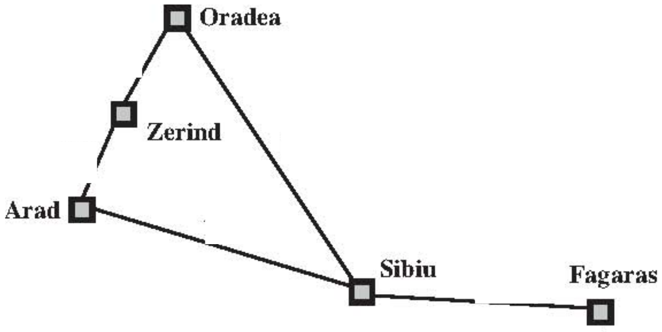


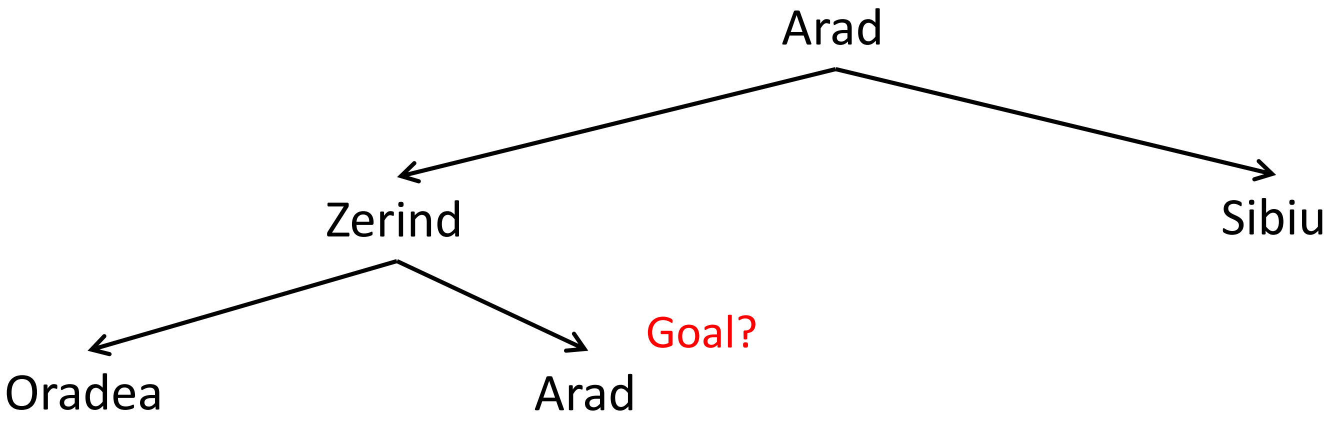
Breadth-first search



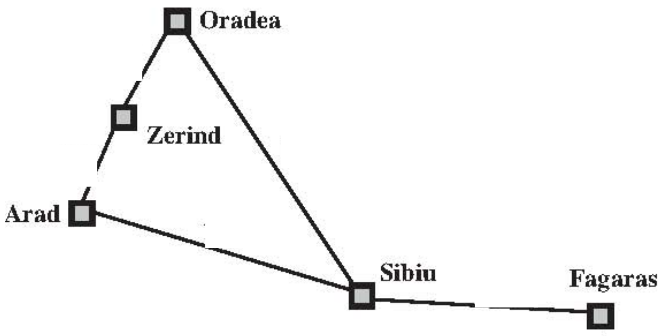


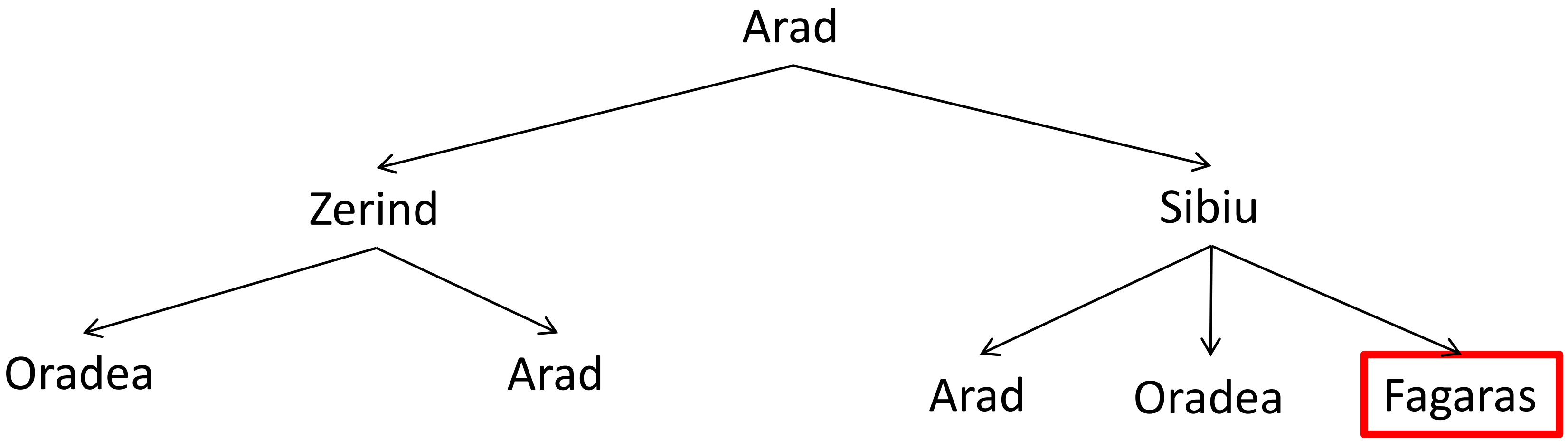
Breadth-first search



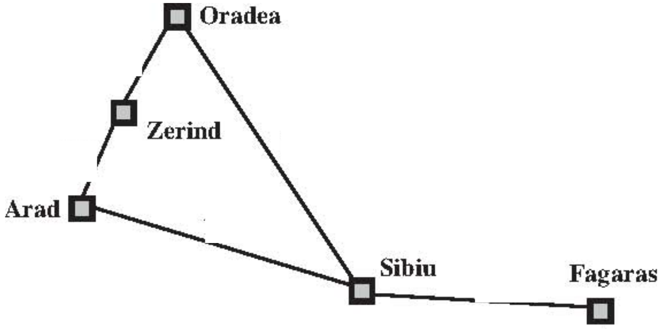


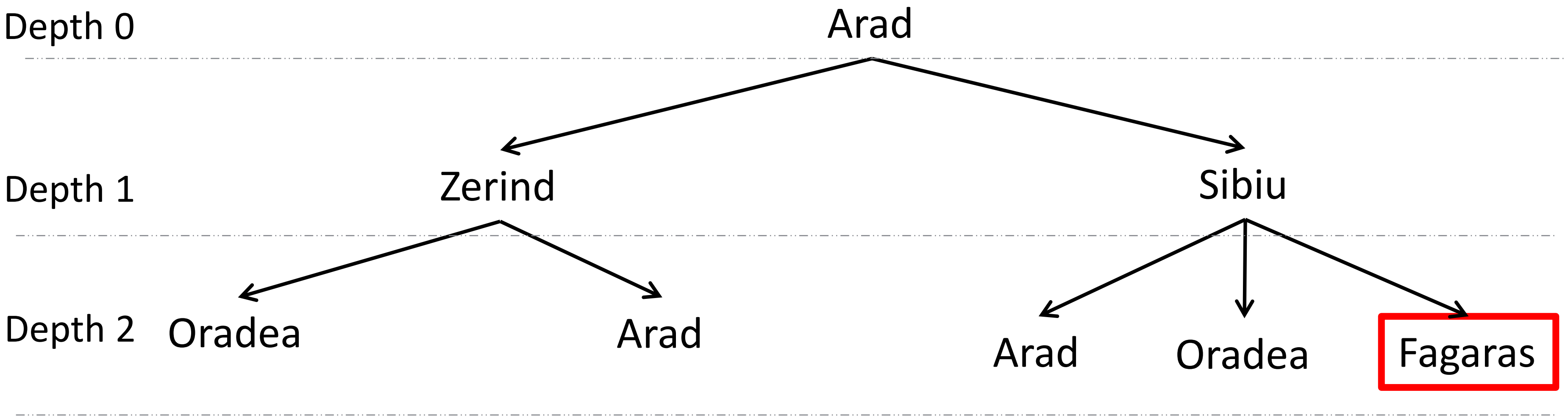
Breadth-first search





Breadth-first search





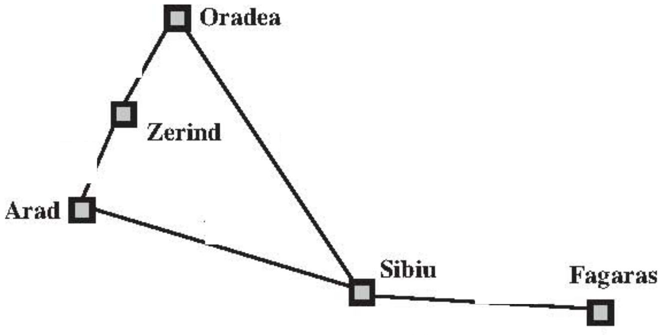
8 nodes explored

Path: Arad-Sibiu-Fagaras (shortest path possible)

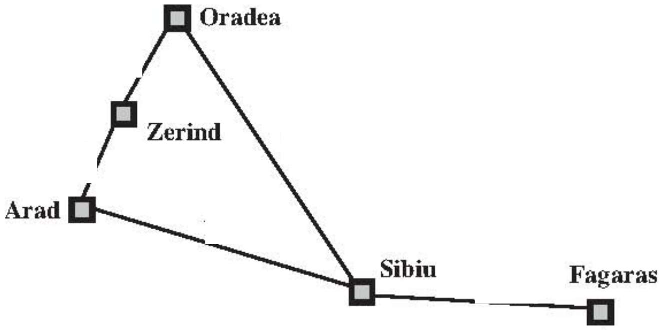
Breadth first search

```
function BFS(G, root,  goal)    // G is your representation problem
{
  queue = root                                  //queue initialized with root node
  while queue is not empty
 {
  current = Pop(queue)                          //removes first element from queue
   if current is the goal  then return current
     else
      {
        for each child n of current             // get from representation of G
        Add_end (n, queue)                      // add at the end of the queue
       }
    }
}
```

Depth-first search



Arad

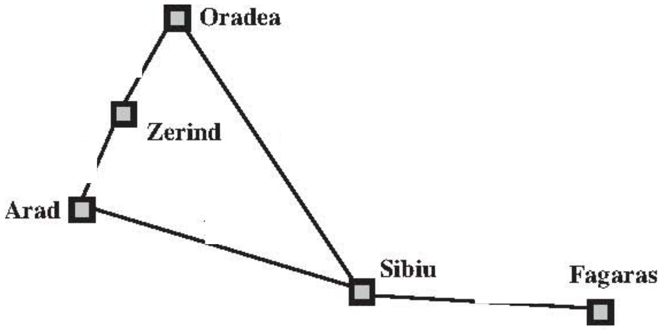


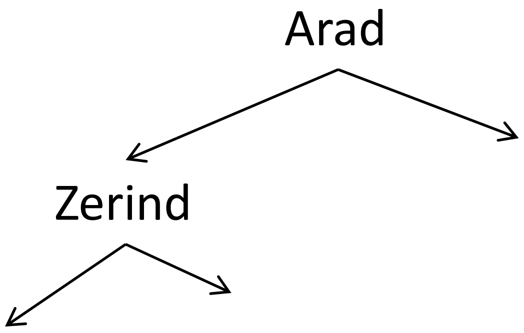
Depth-first search

Arad

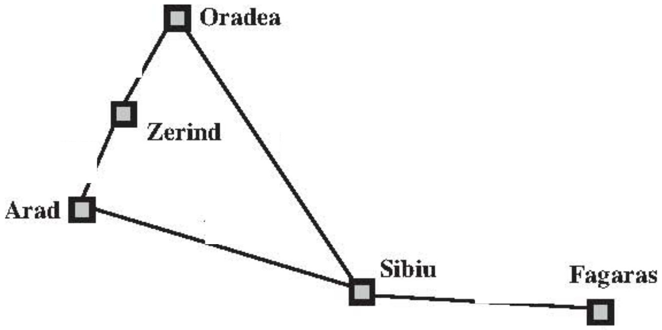
Zerind

Depth-first search





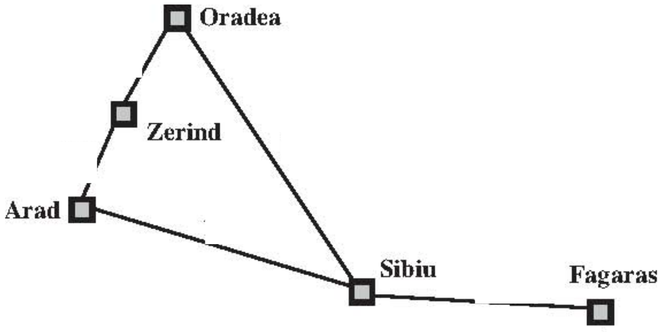
Depth-first search



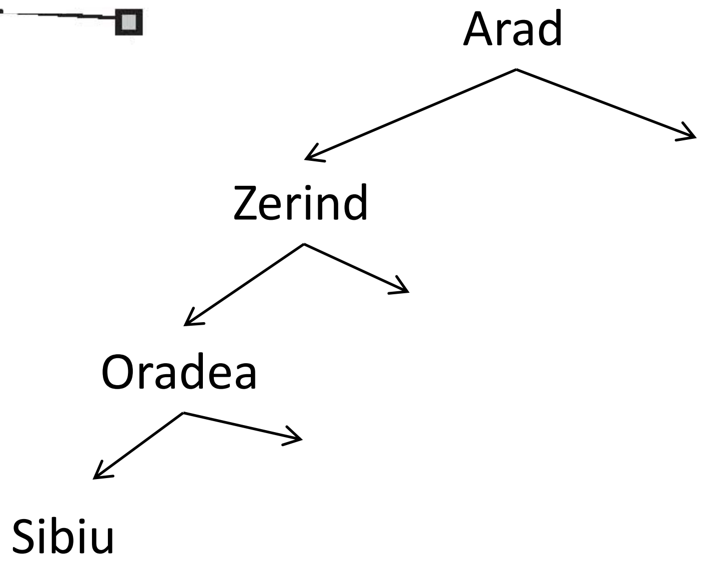
Arad

Zerind

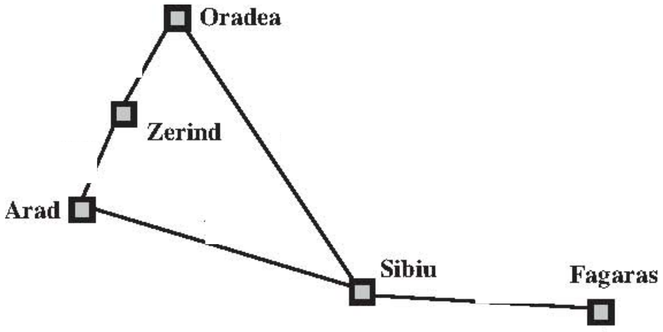
Oradea

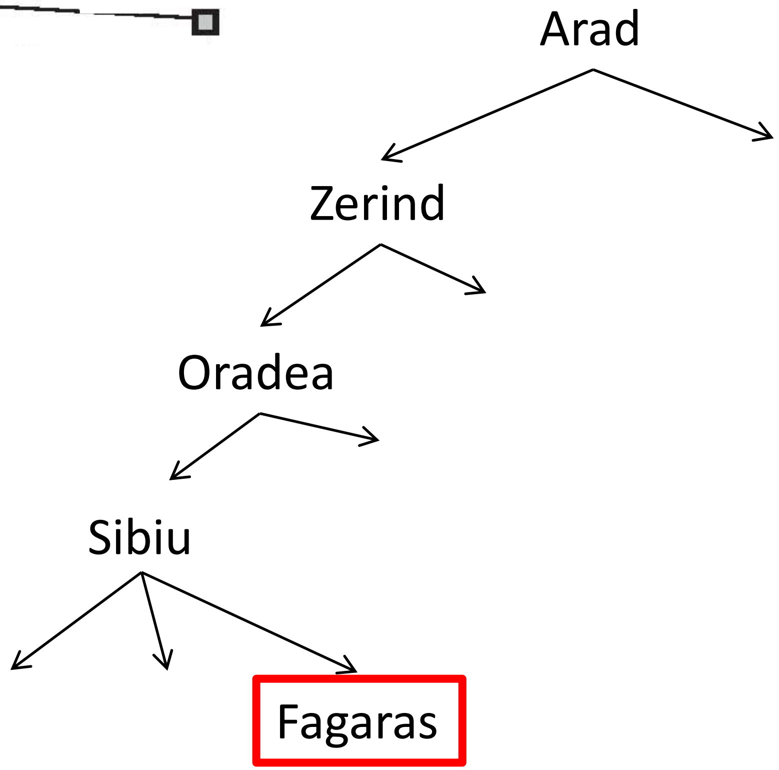


Depth-first search

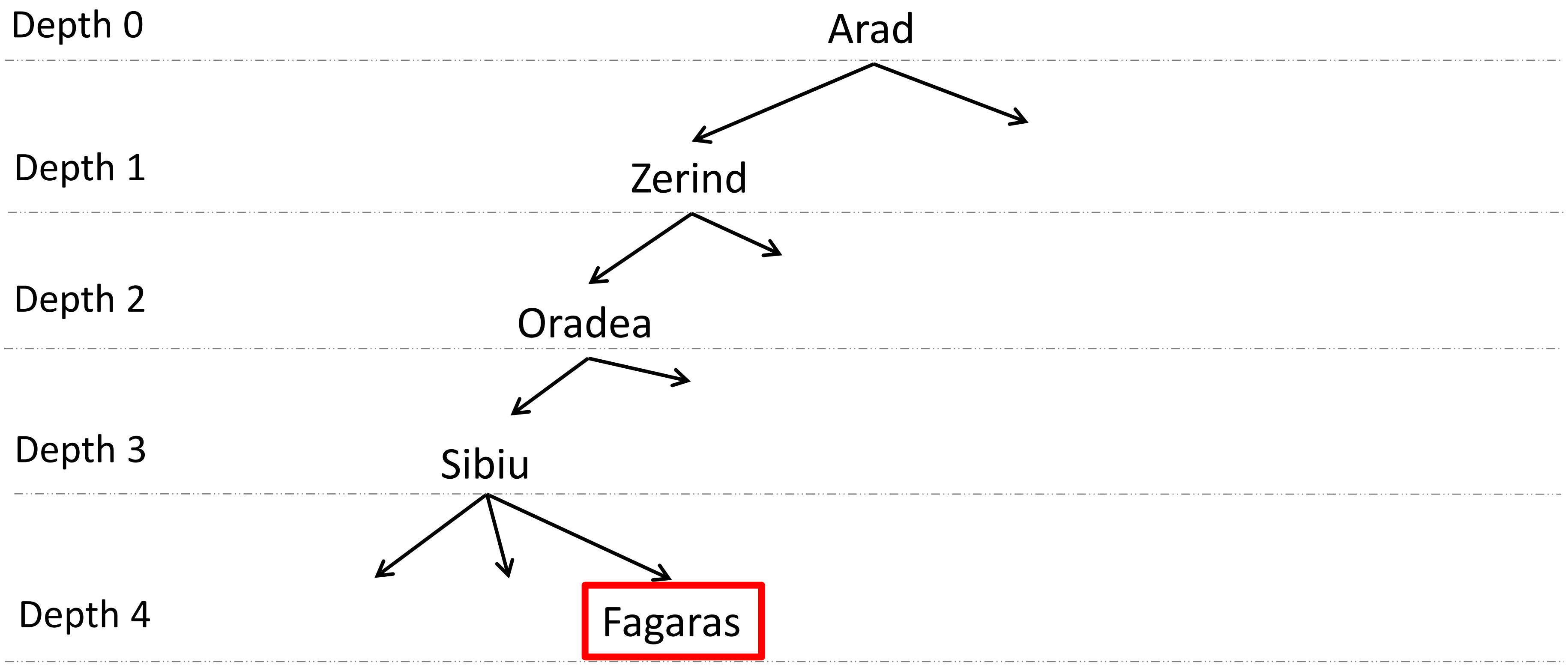


Depth-first search





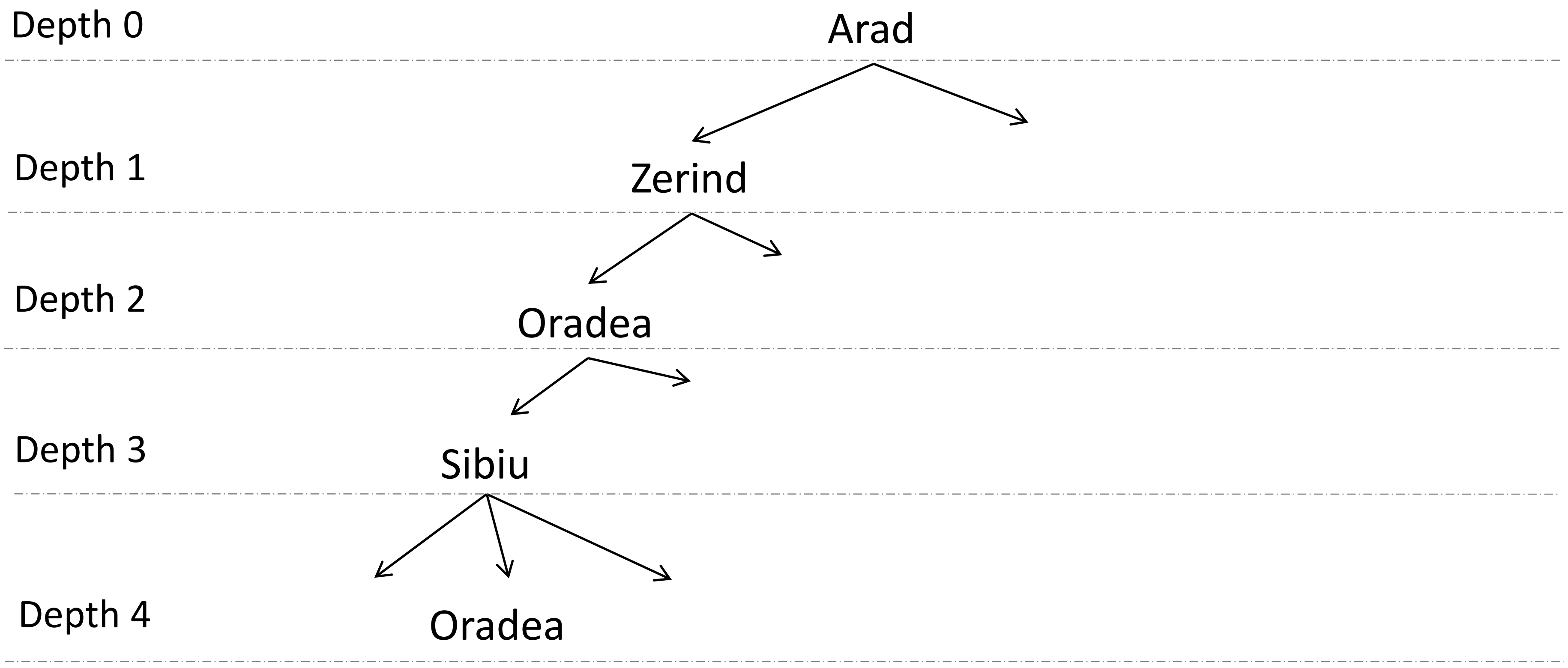
Depth-first search



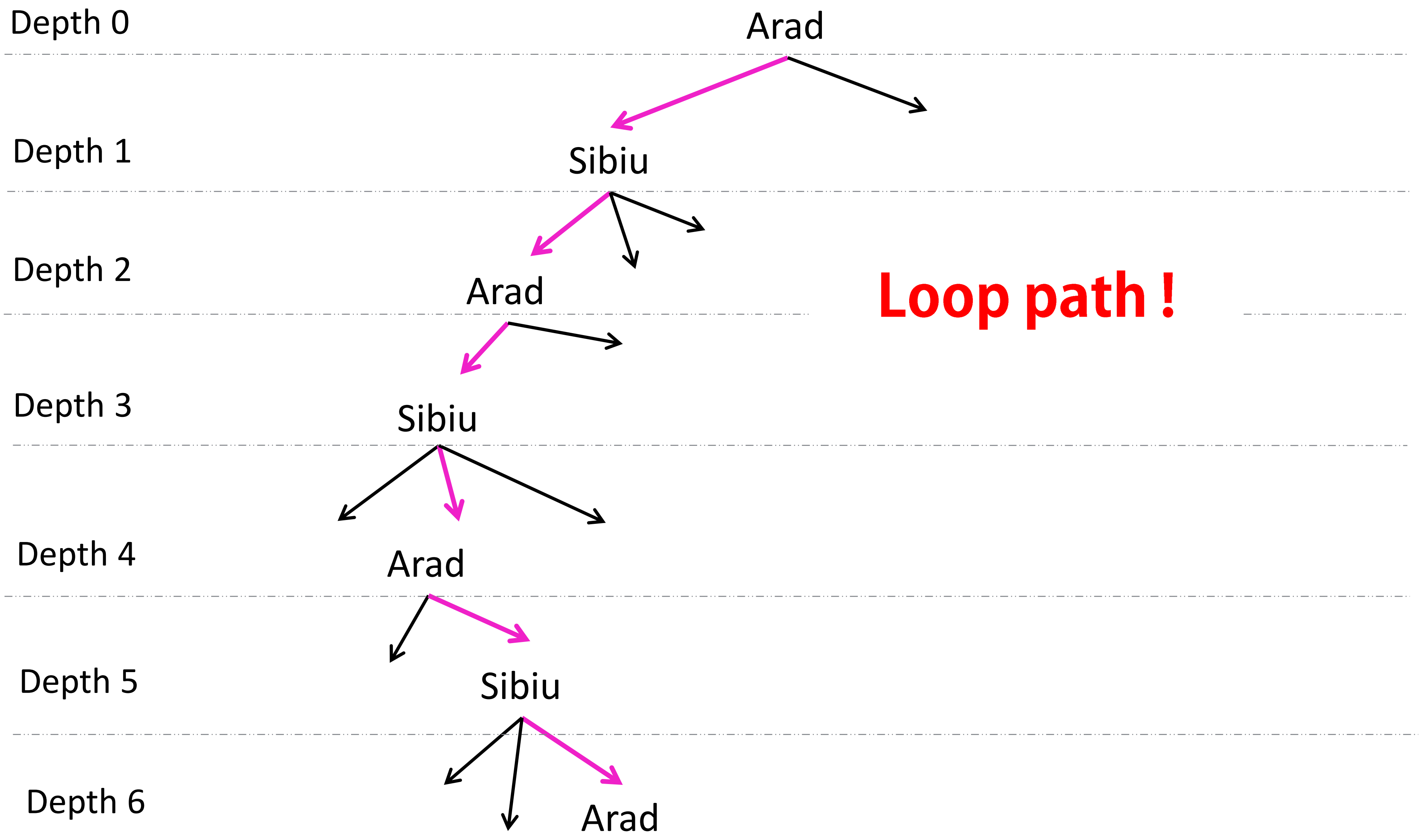
5 nodes explored

Path: Arad – Zerind – Oradea – Sibiu- Fagaras

Depth-first search



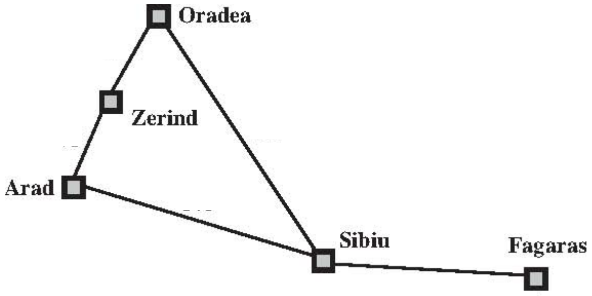
Depth-first search



Depth first search

```
function DFS(G, root,  goal)    // G is your representation problem
{
  queue = root                                  //queue initialized with root node
  while queue is not empty
 {
  current = Pop(queue)                          //removes first element from queue
   if current is the goal  then return current
     else
      {
        for each child n of current             // get from representation of G
        Add_front (n, queue)                     // add at the beginning of the queue
       }
    }
}
```

Depth-limited search



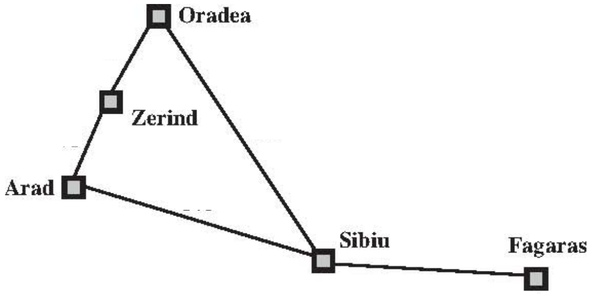
Limit = 3

Depth 0

Arad

Depth 1

Zerind



Depth-limited search

Limit = 3

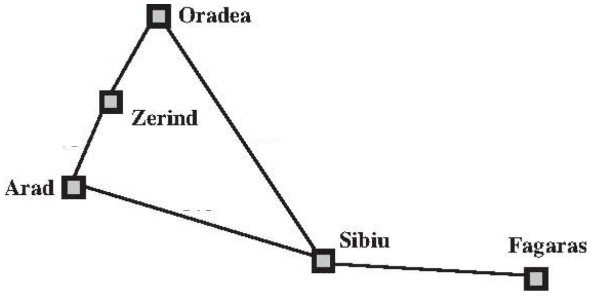
Depth 0 Arad

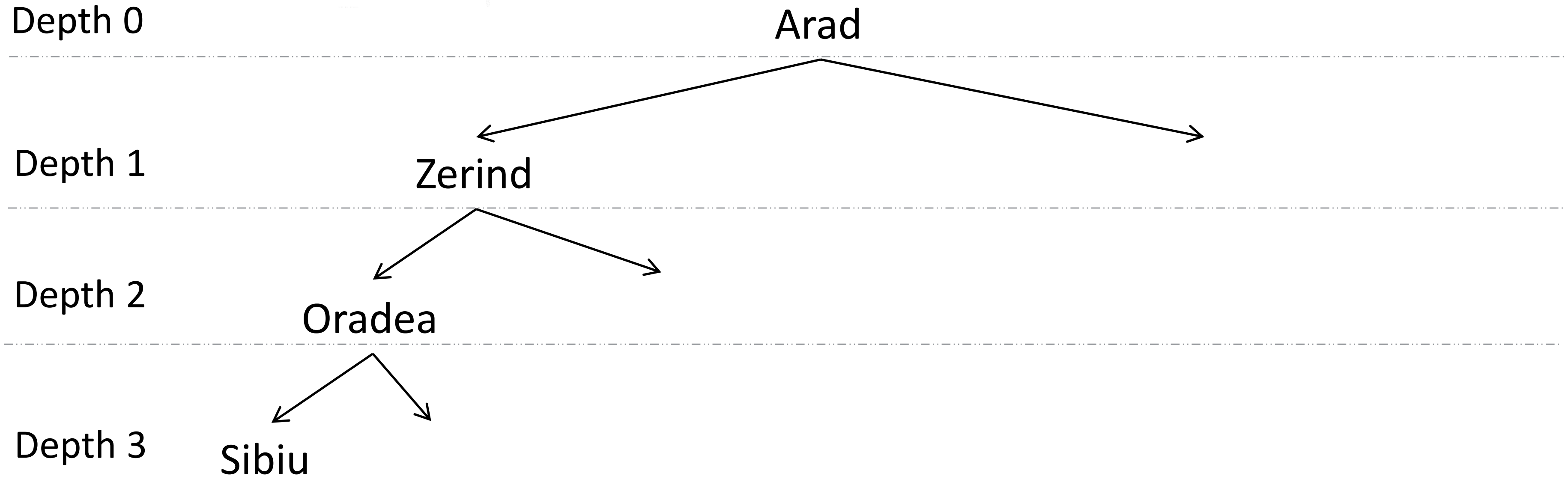
Depth 1 Zerind

Depth 2 Oradea

Depth-limited search

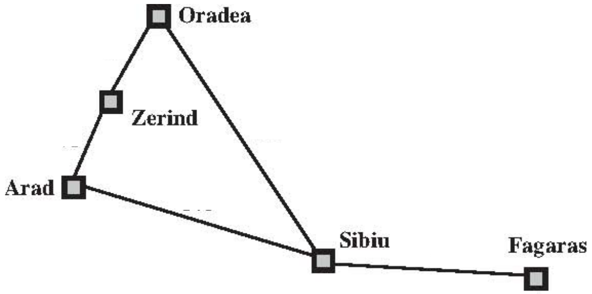
Limit = 3





Depth-limited search

Limit = 3

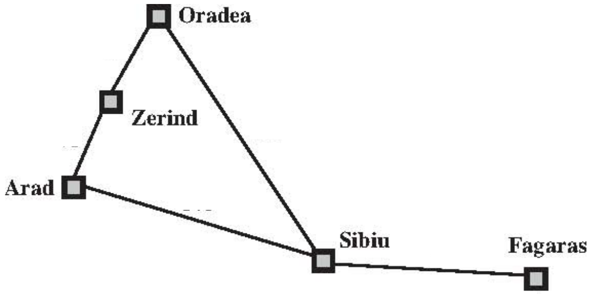


Depth 0 Arad

Depth 1 Zerind

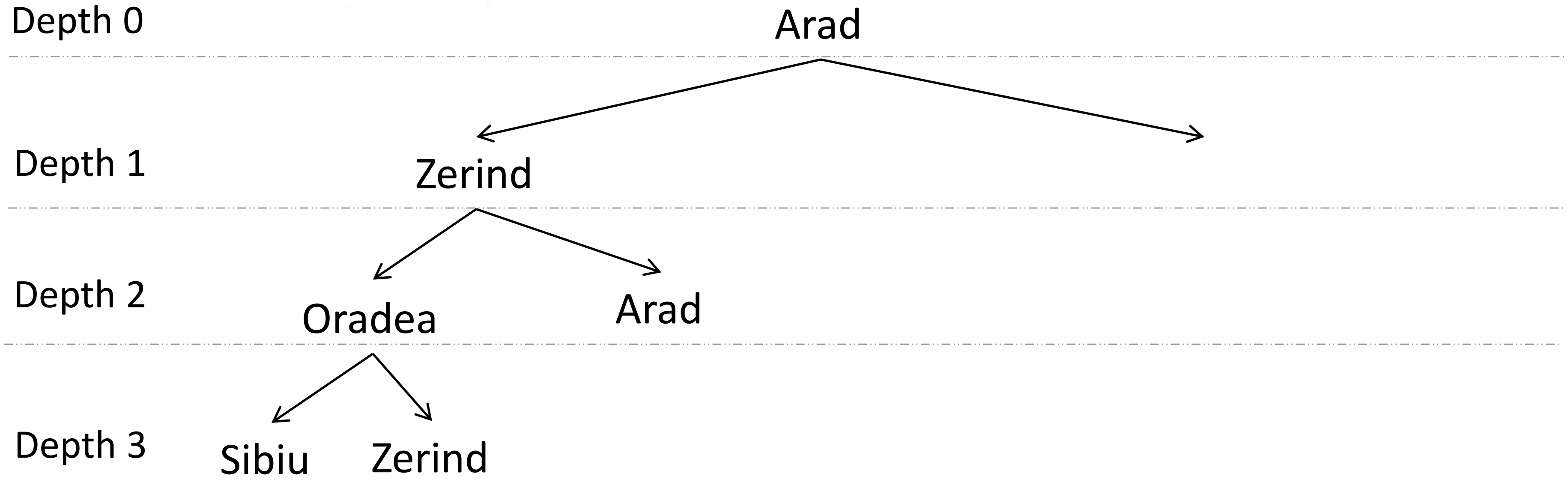
Depth 2 Oradea

Depth 3 Sibiu Zerind

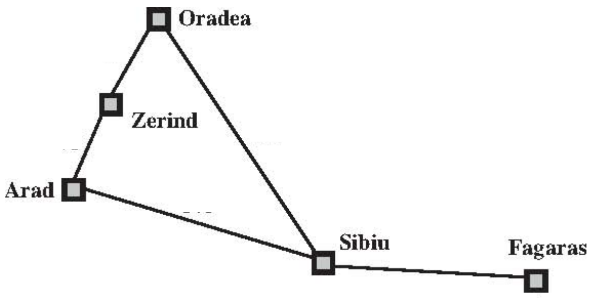


Depth-limited search

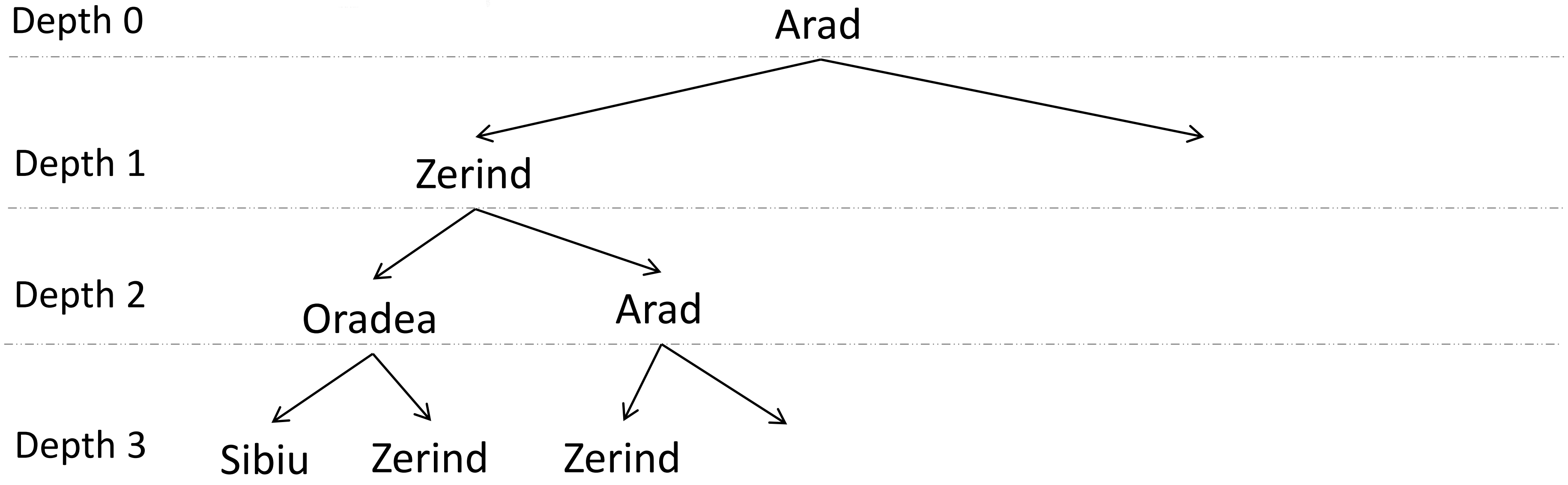
Limit = 3



Depth-limited search

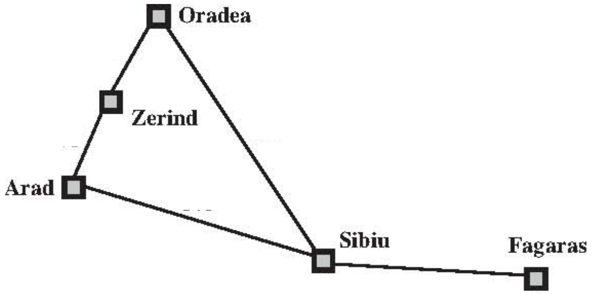


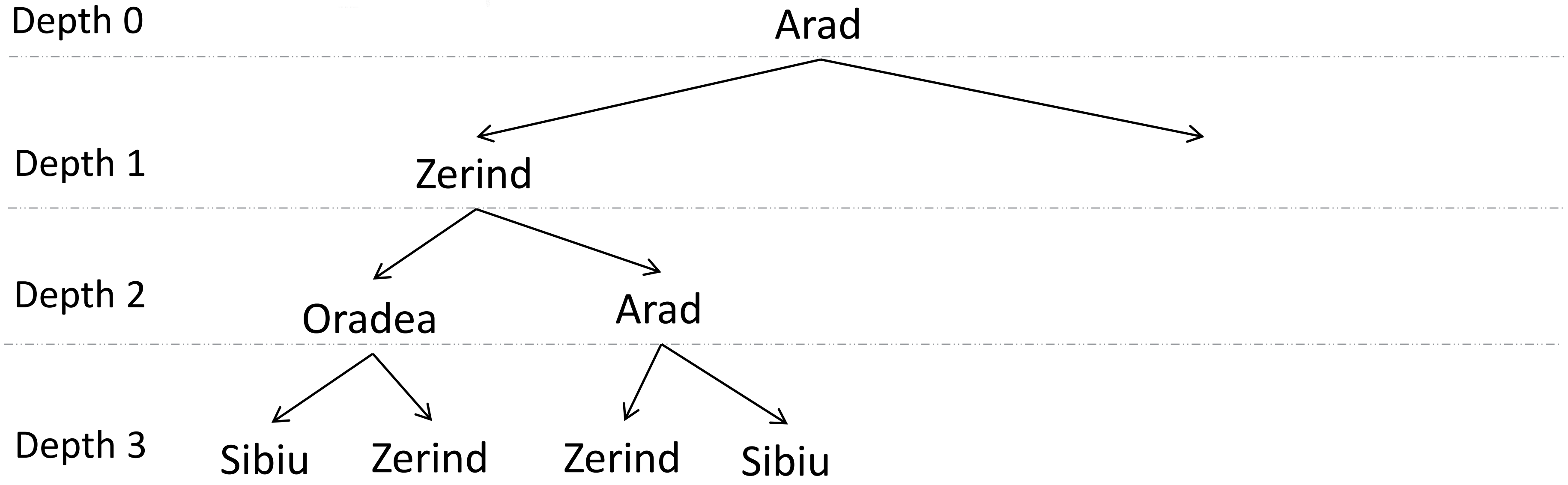
Limit = 3



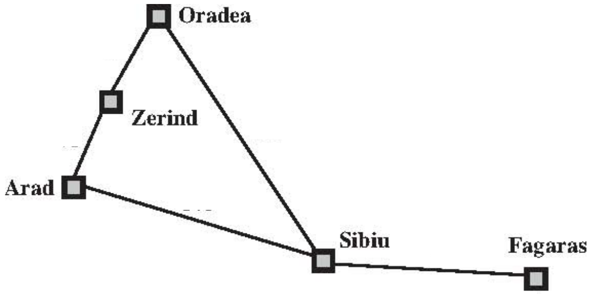
Depth-limited search

Limit = 3

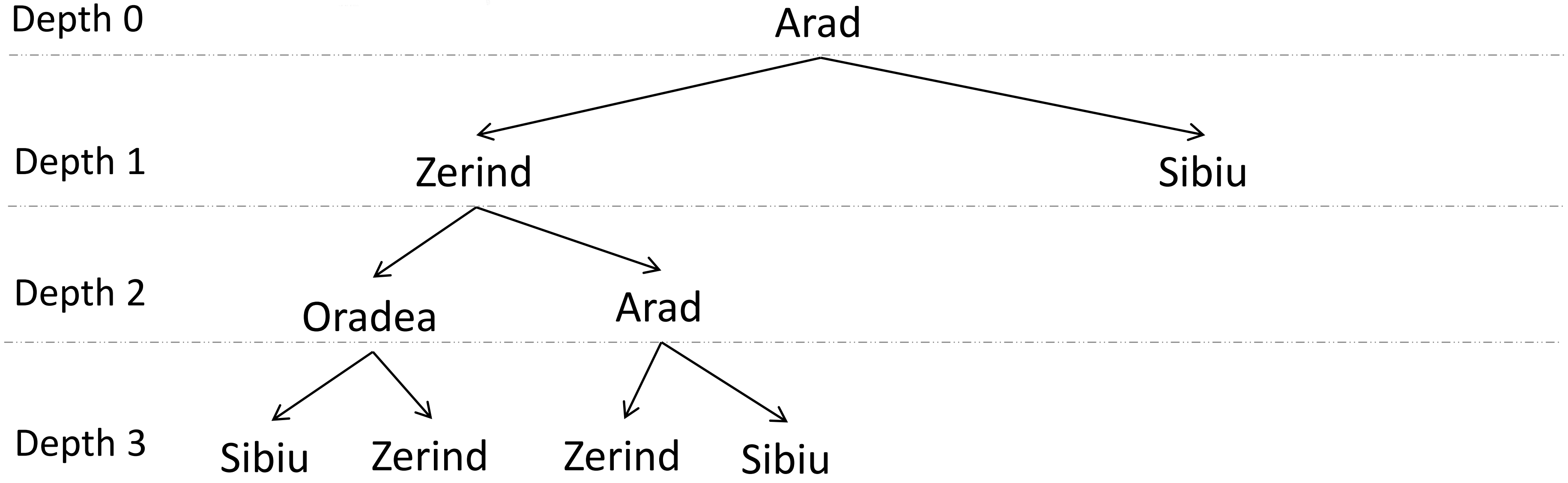




Depth-limited search

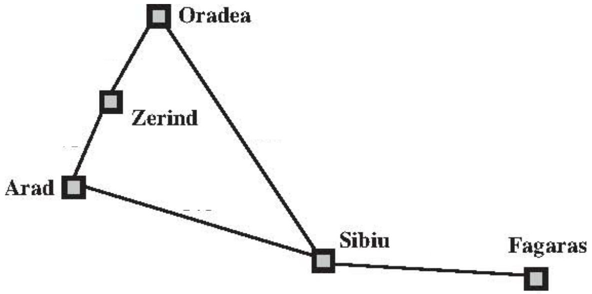


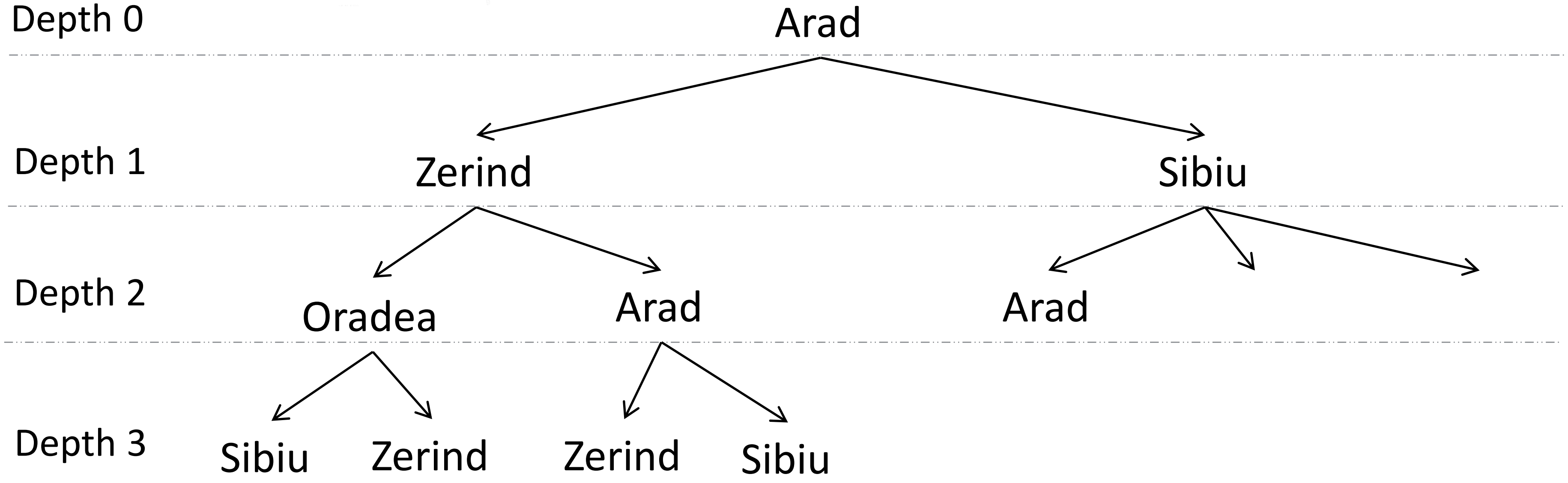
Limit = 3

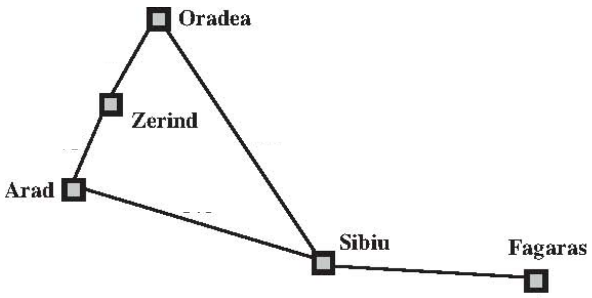


Depth-limited search

Limit = 3

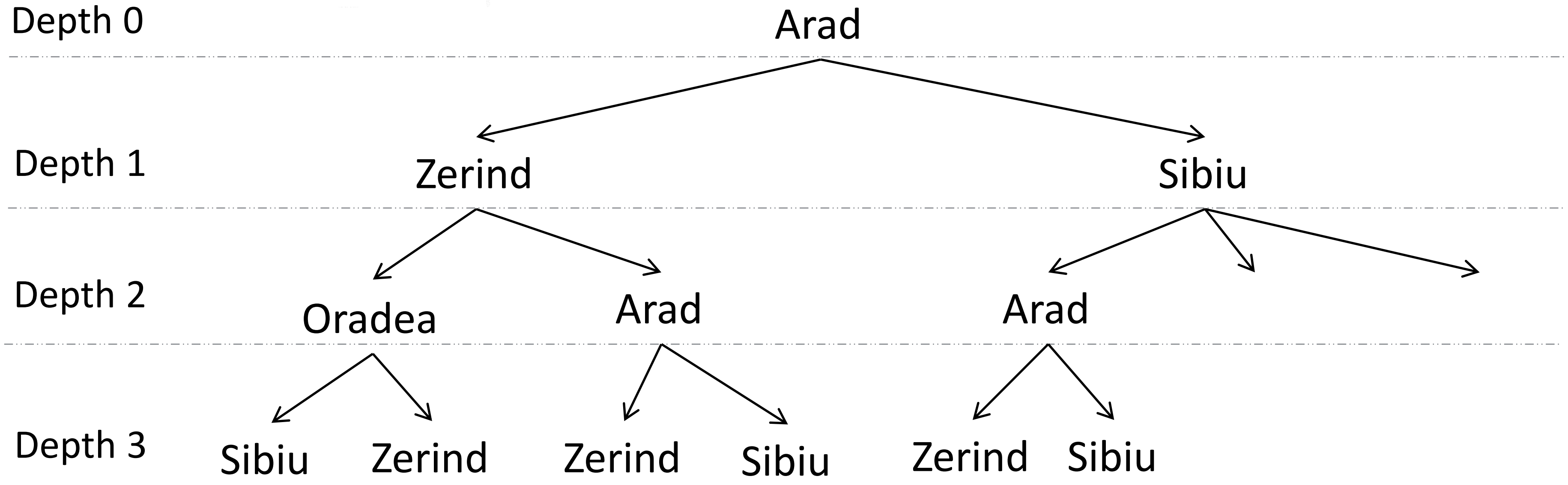




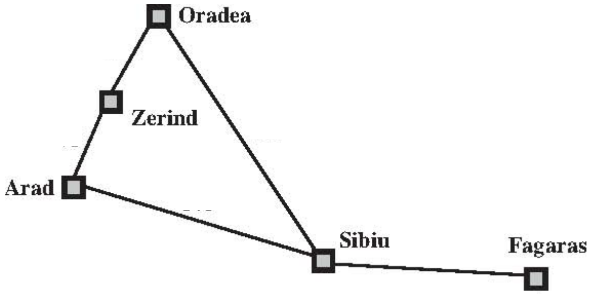


Depth-limited search

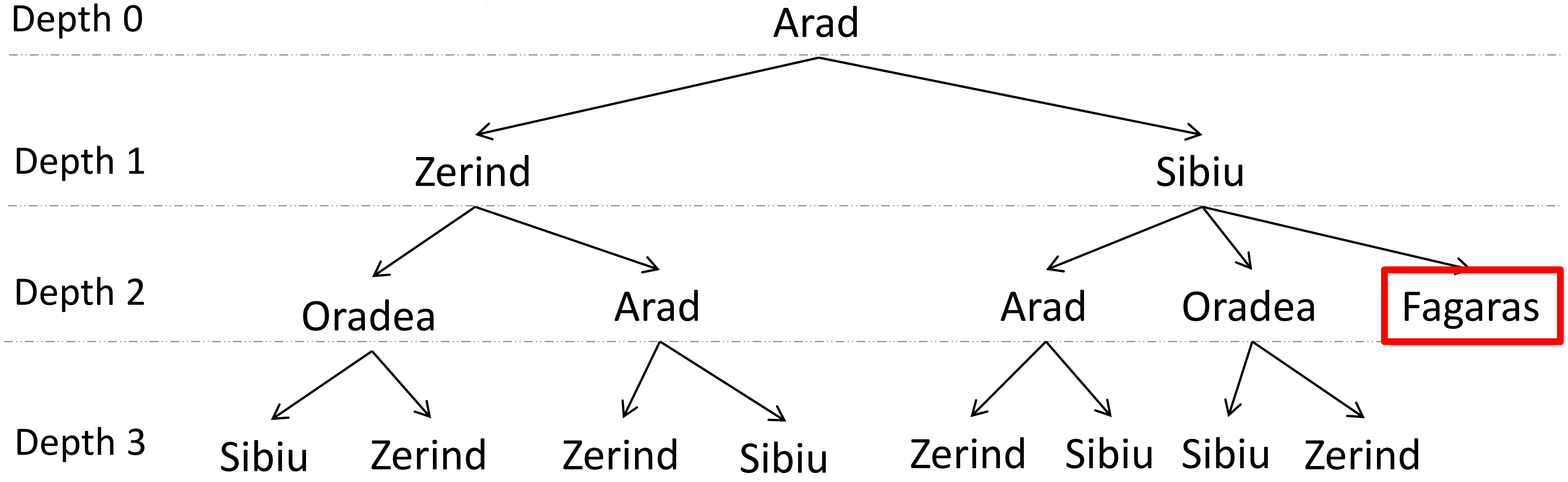
Limit = 3



Depth-limited search



Limit = 3

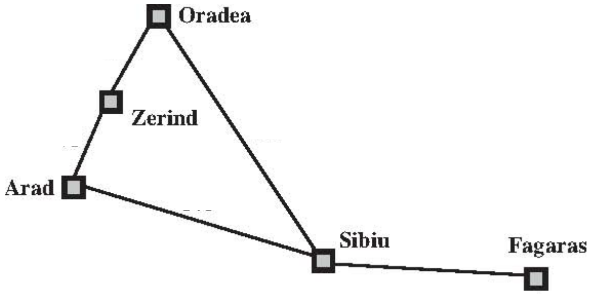


16 nodes explored

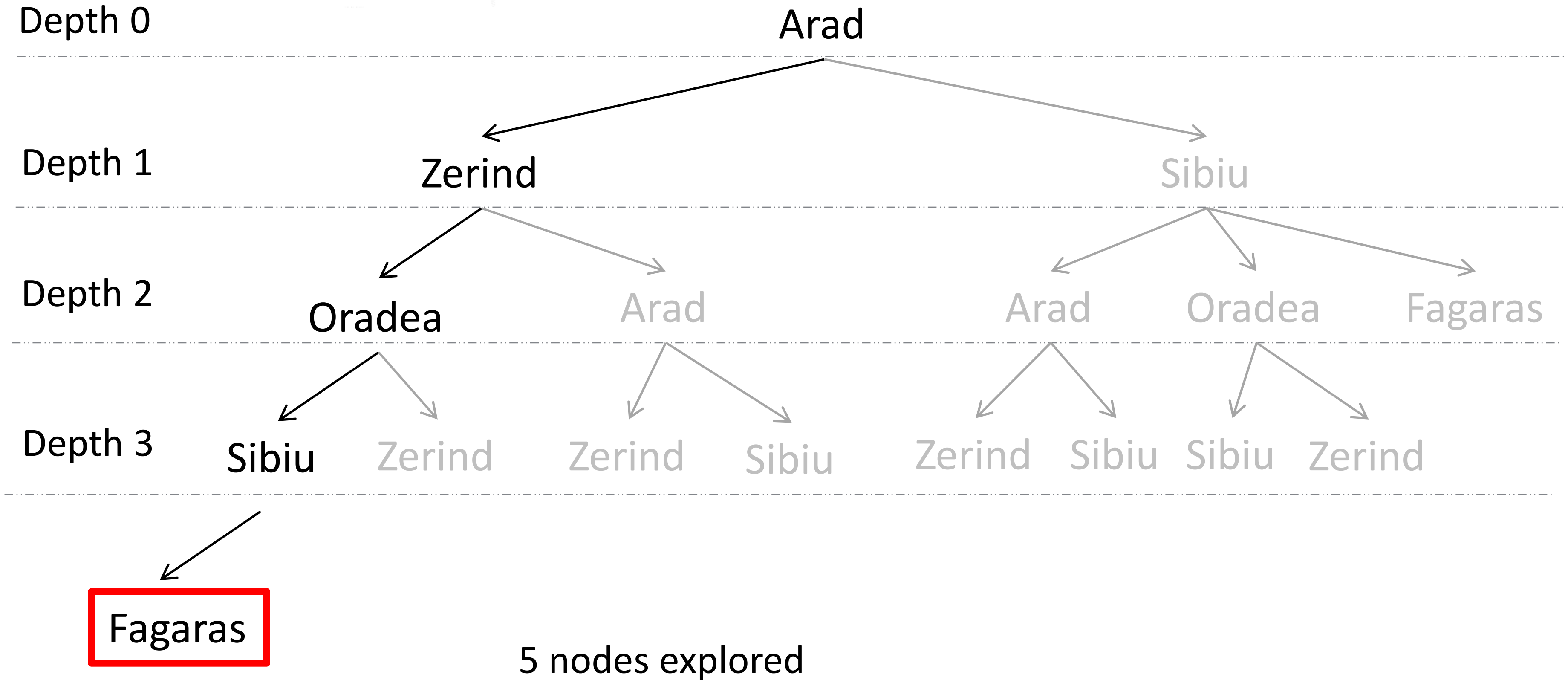
Path: Arad – Sibiu – Fagaras

Shortest path, in this case.

Depth-limited search



Limit = 4



5 nodes explored

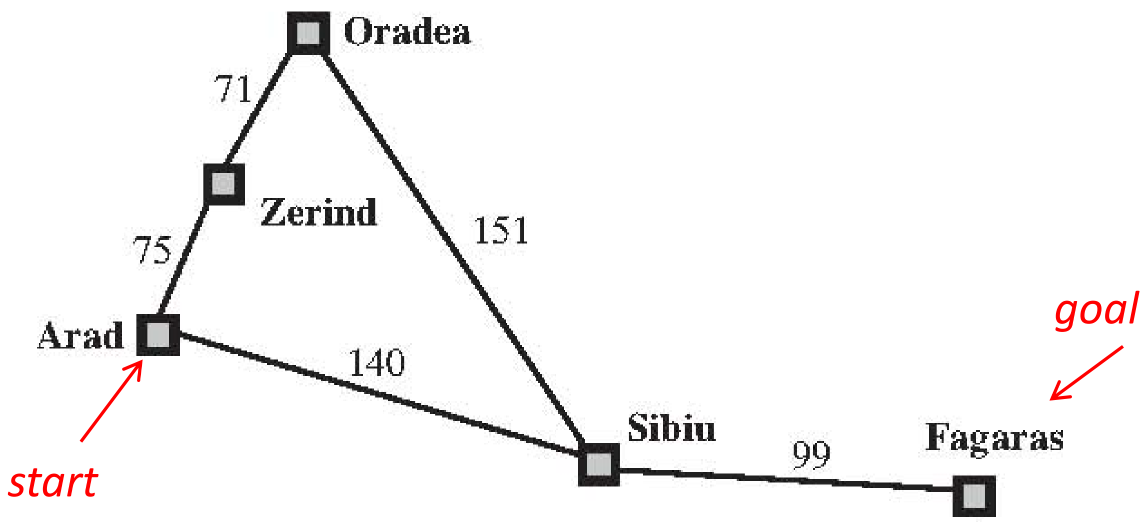
Path: Arad – Zerind – Oradea - Sibiu – Fagaras

Not the shortest path, in this case.

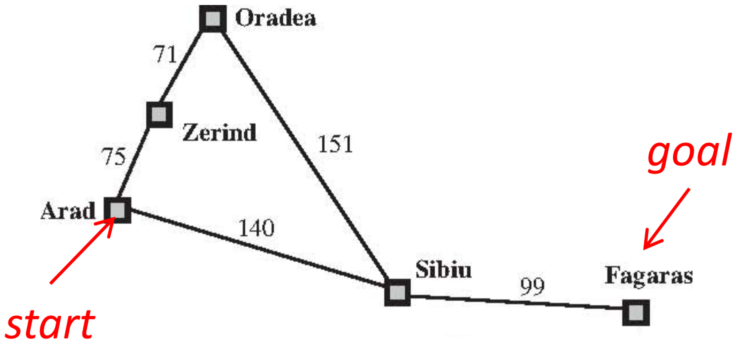
Iterative Deepening

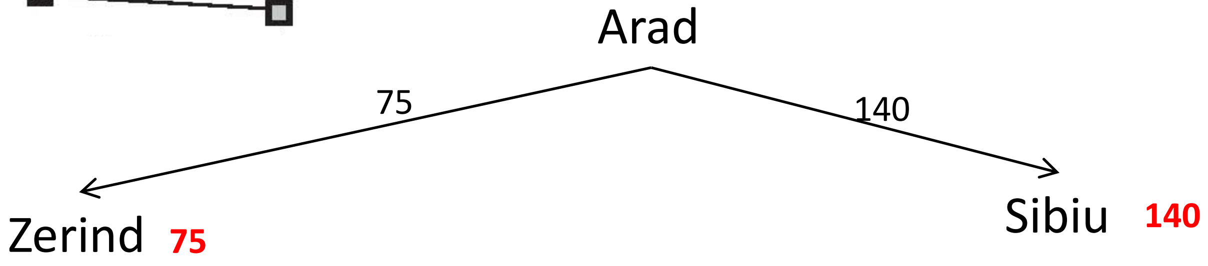
- Several depth limited search with increasing limit
- In depth first search manner search the tree
  - First down to depth 1
  - Secondly down to depth 2
  - Thirdly down to depth 3
  - and so forth until goal state is found
- Each time you start over from the beginning

Uniform-cost search

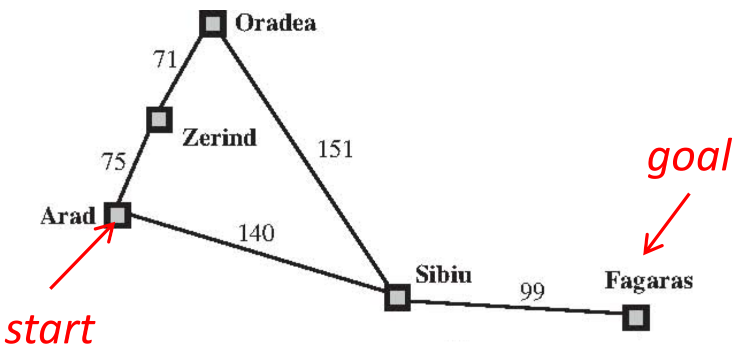


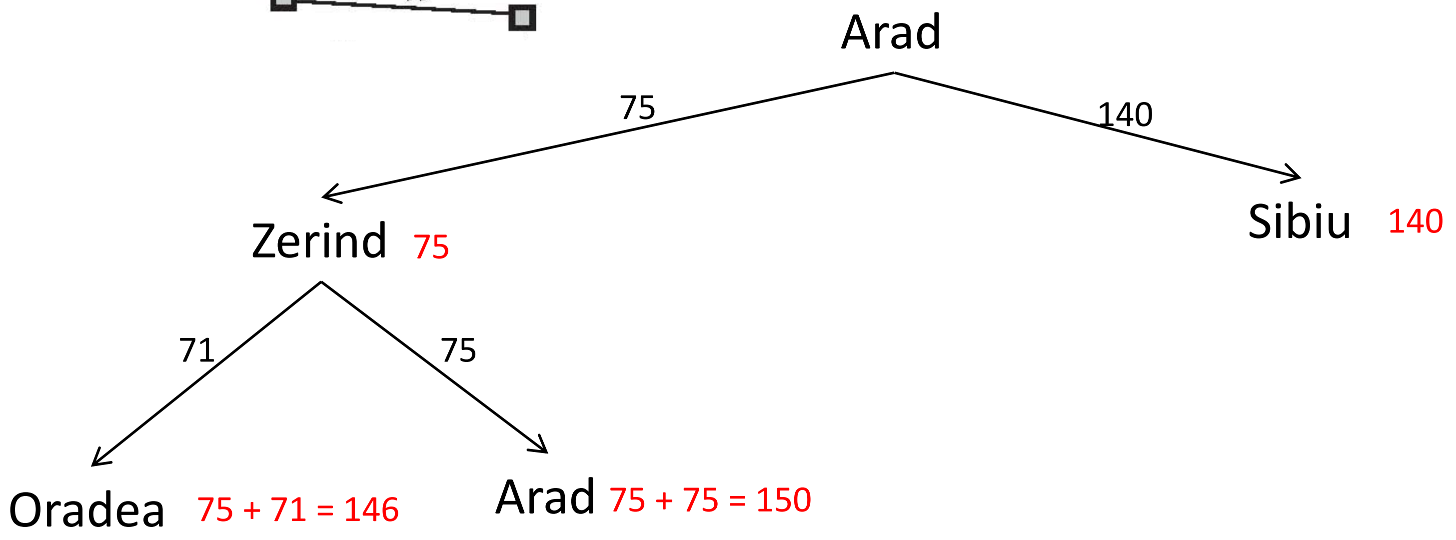
Uniform-cost search



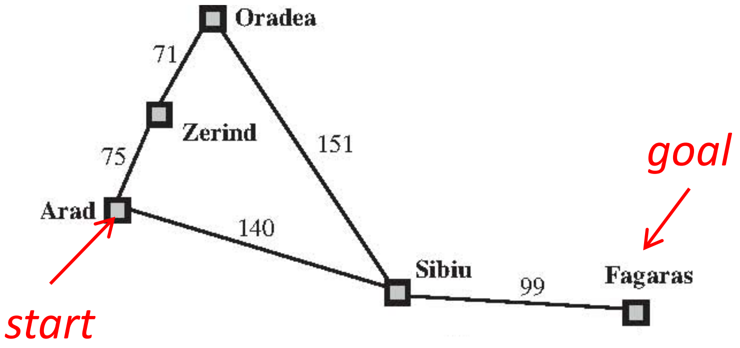


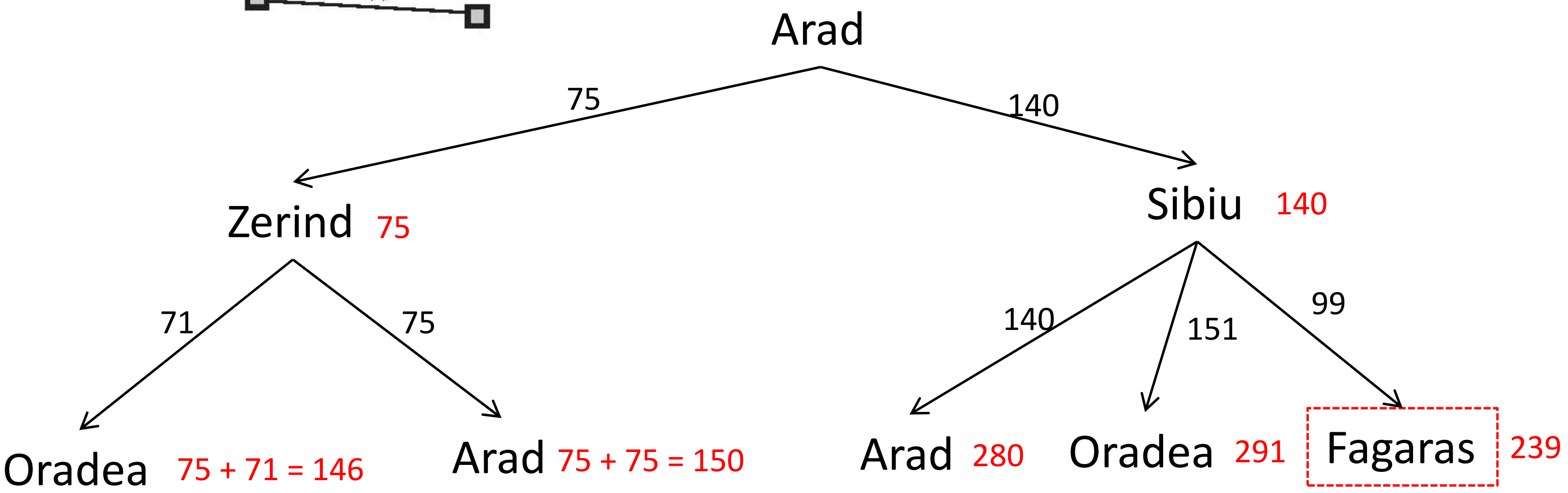
Uniform-cost search



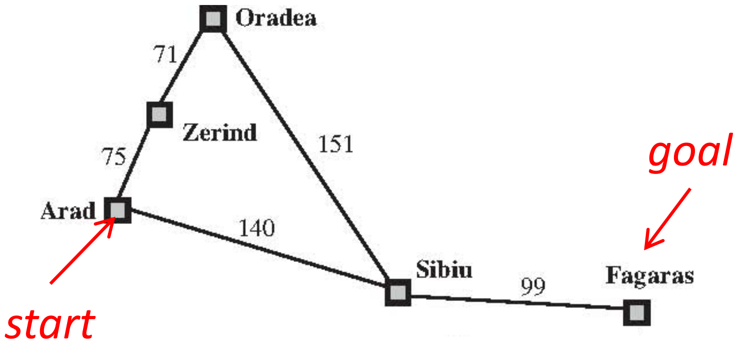


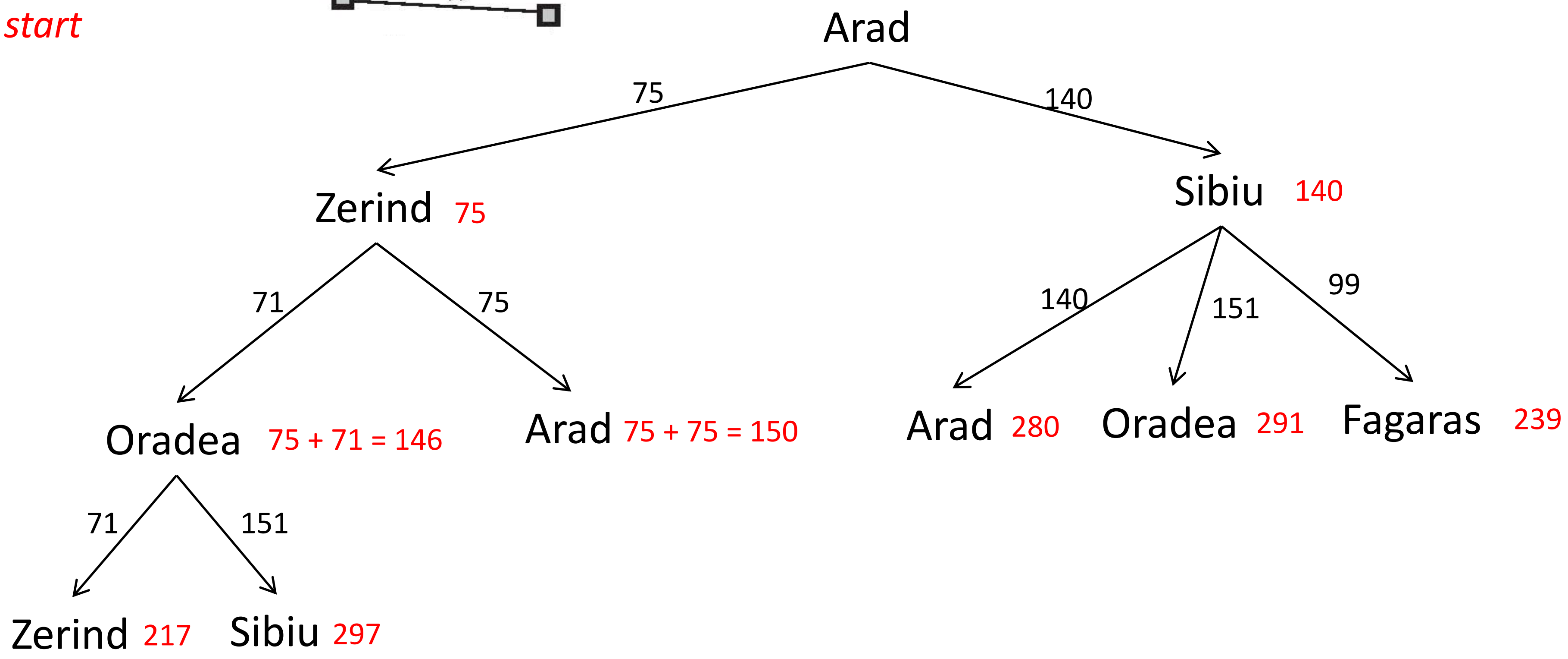
Uniform-cost search



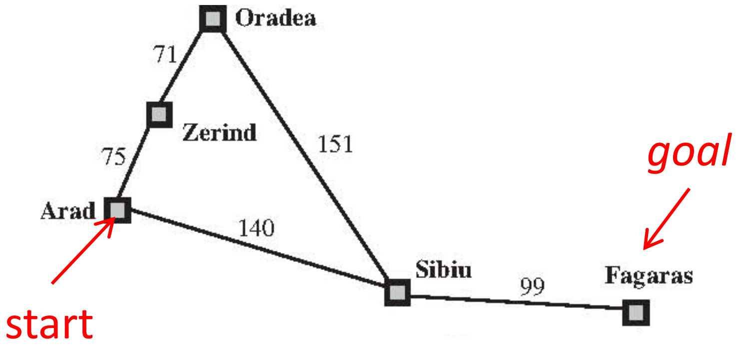


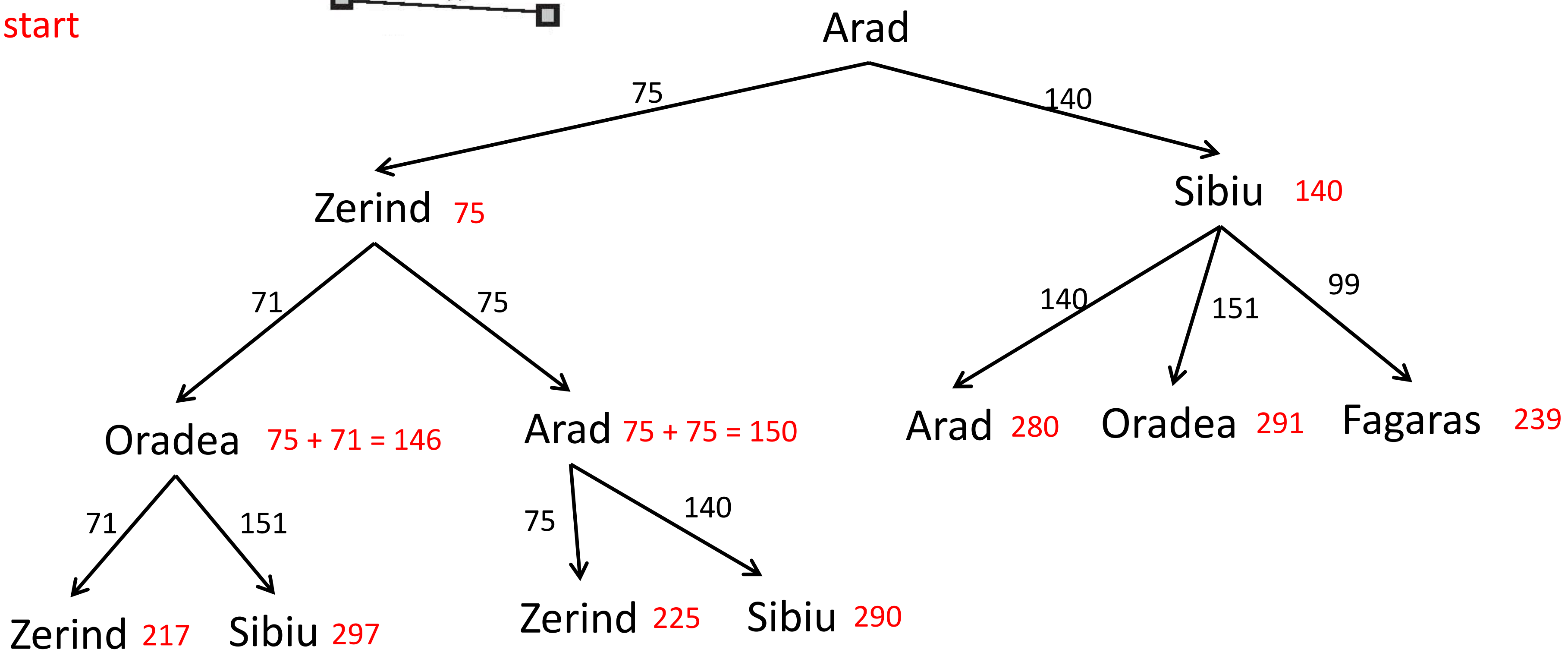
Uniform-cost search



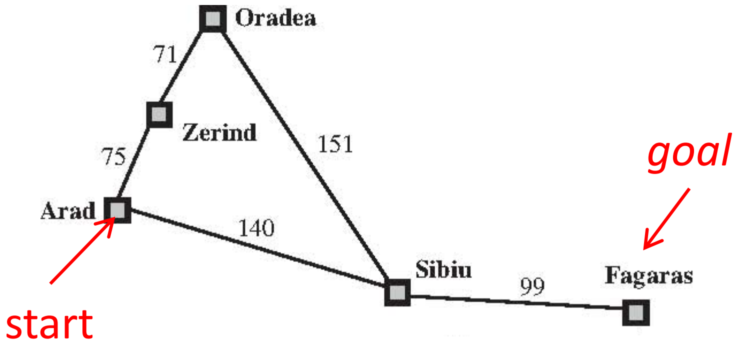


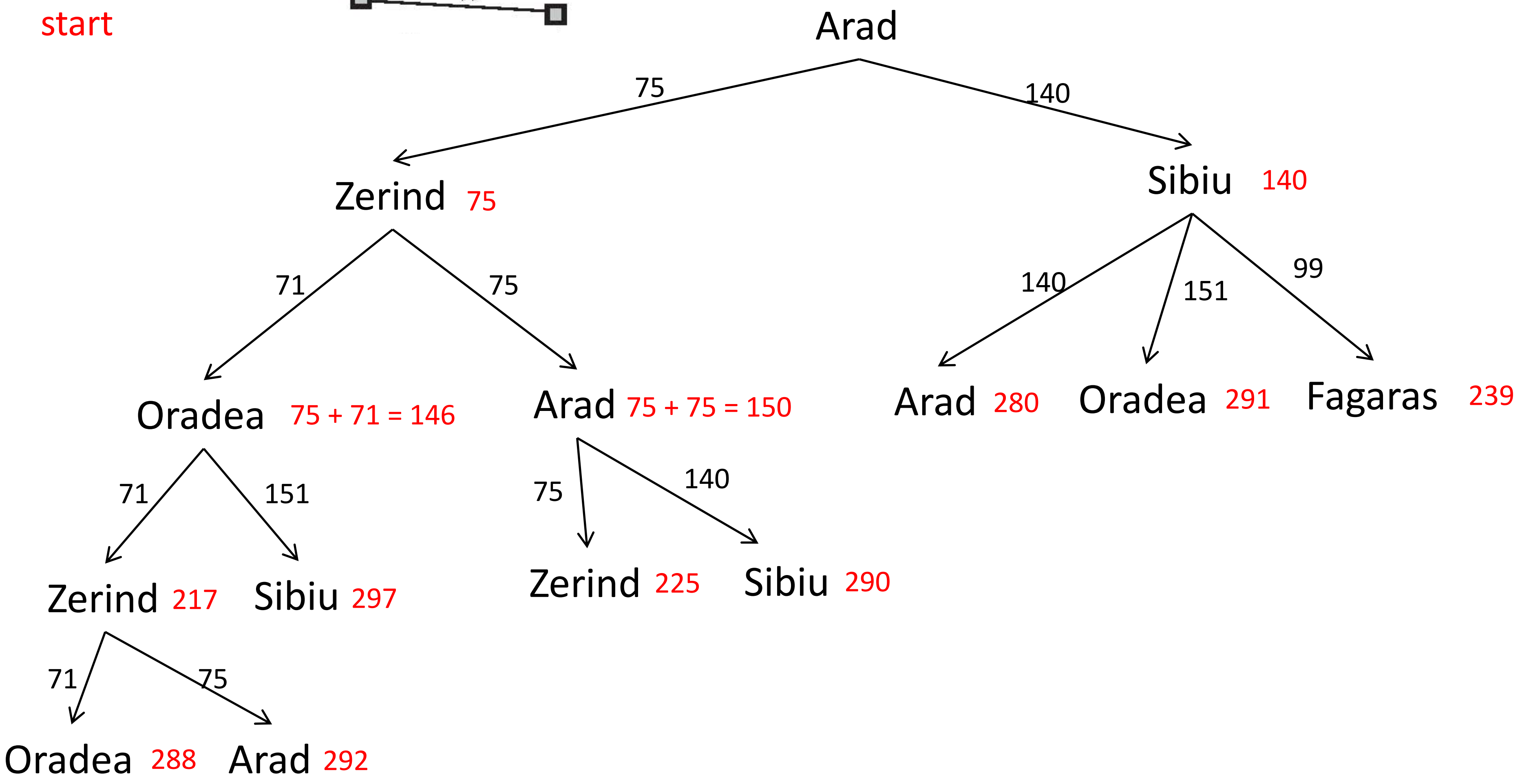
Uniform-cost search



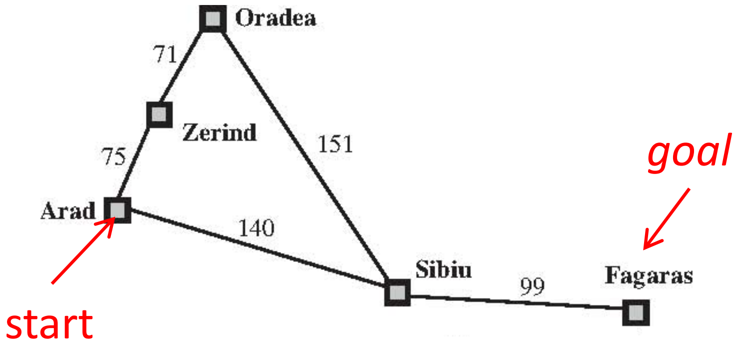


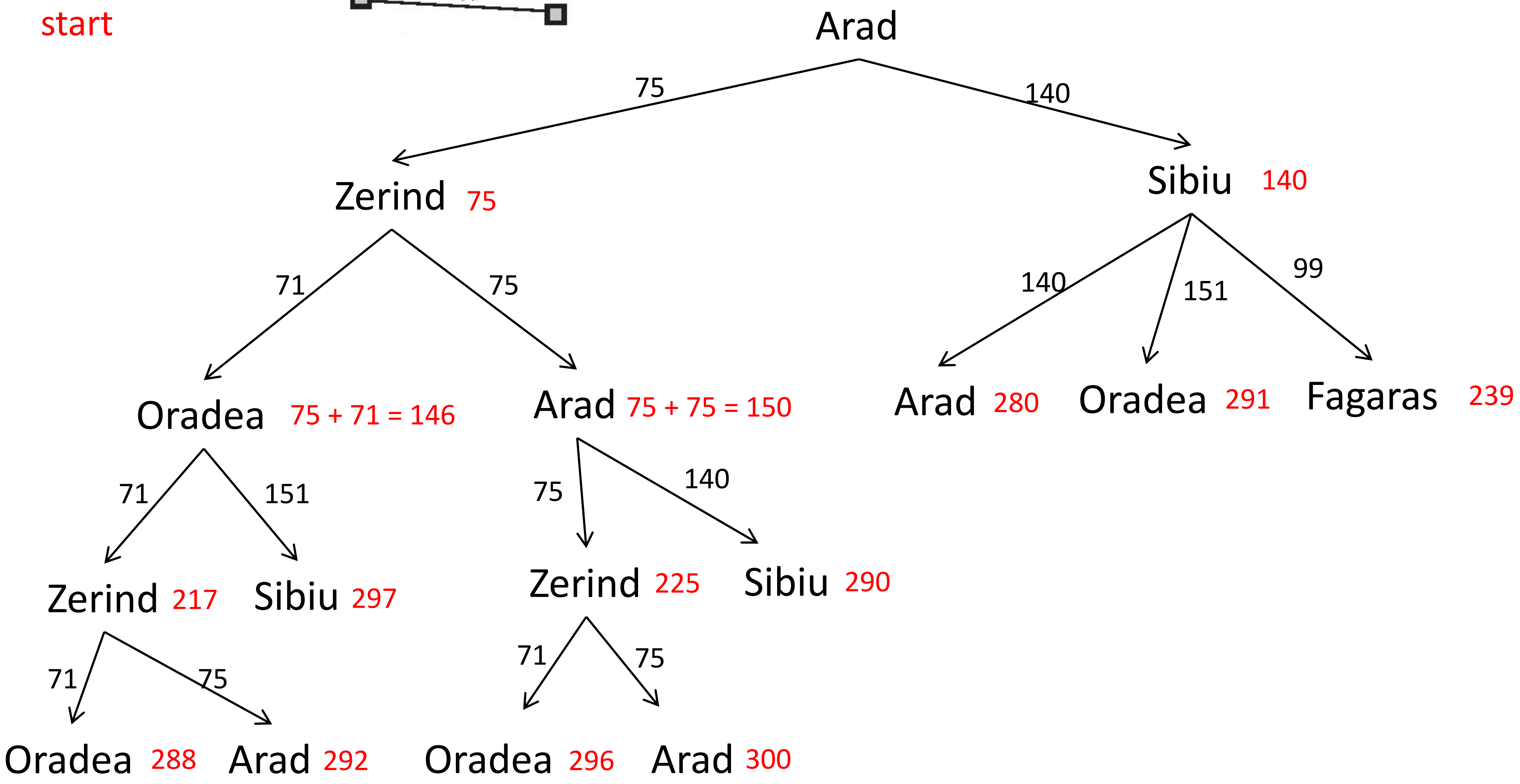
Uniform-cost search



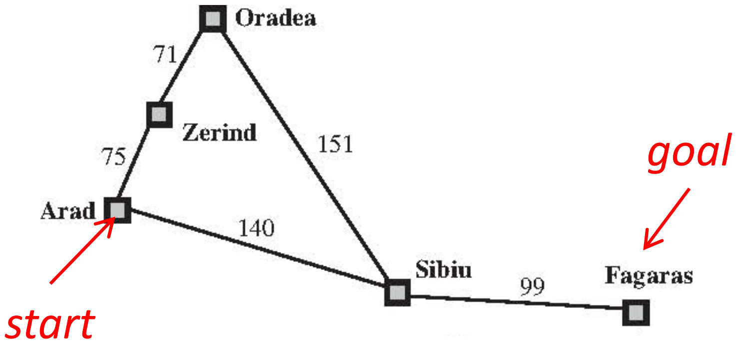


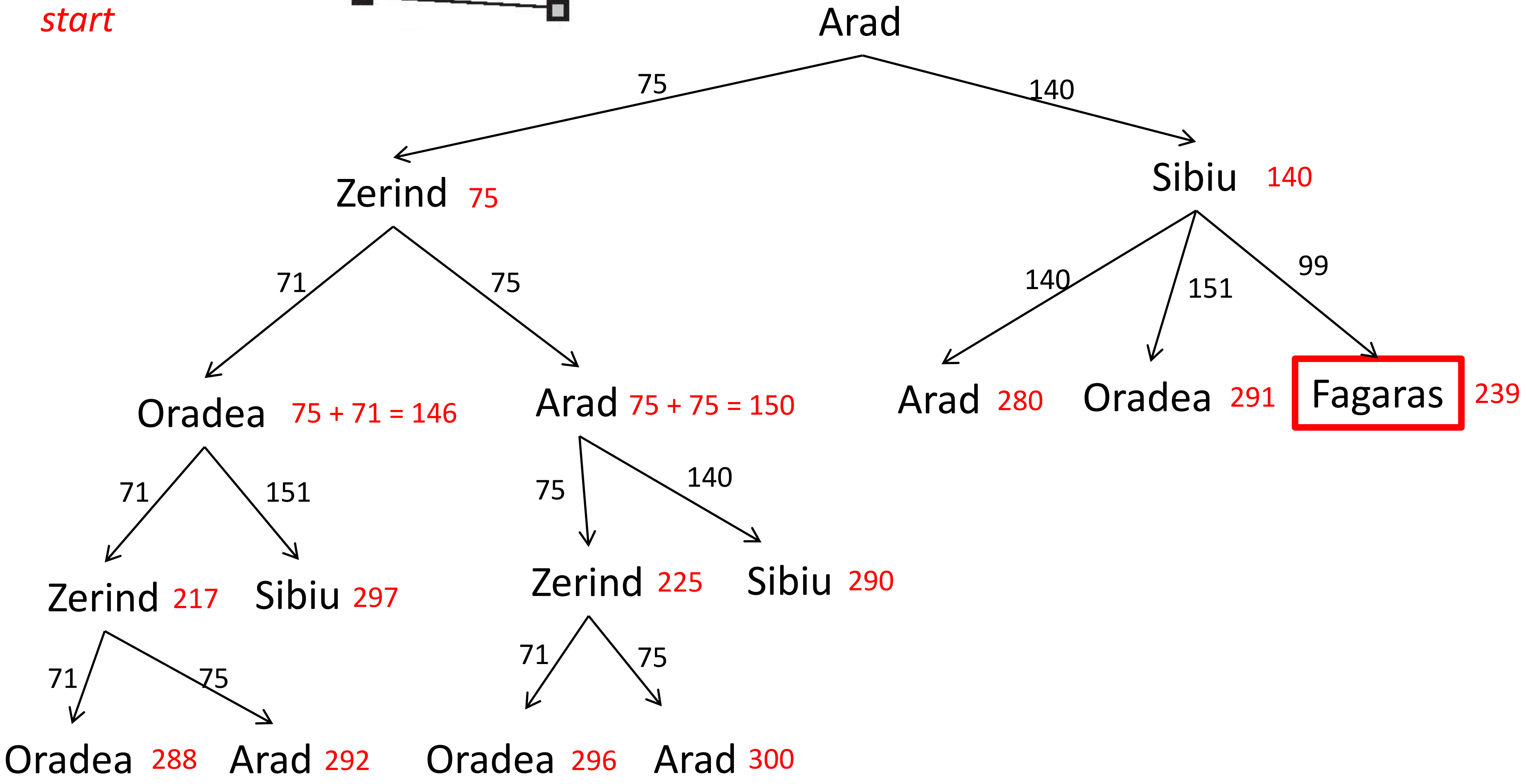
Uniform-cost search





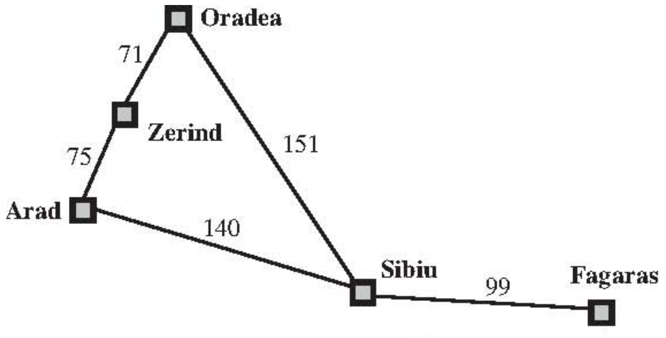
Uniform-cost search

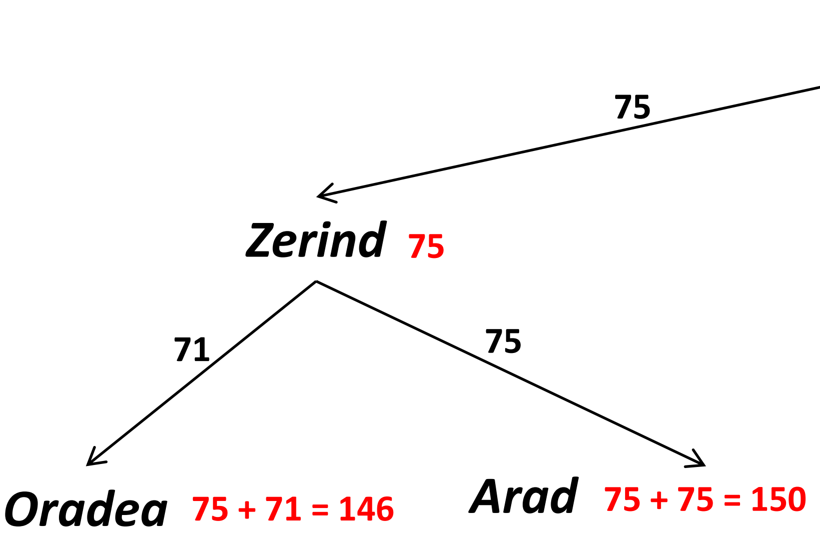




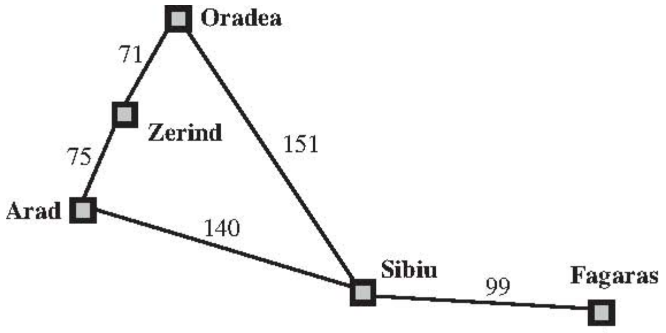
Cost function g(n)

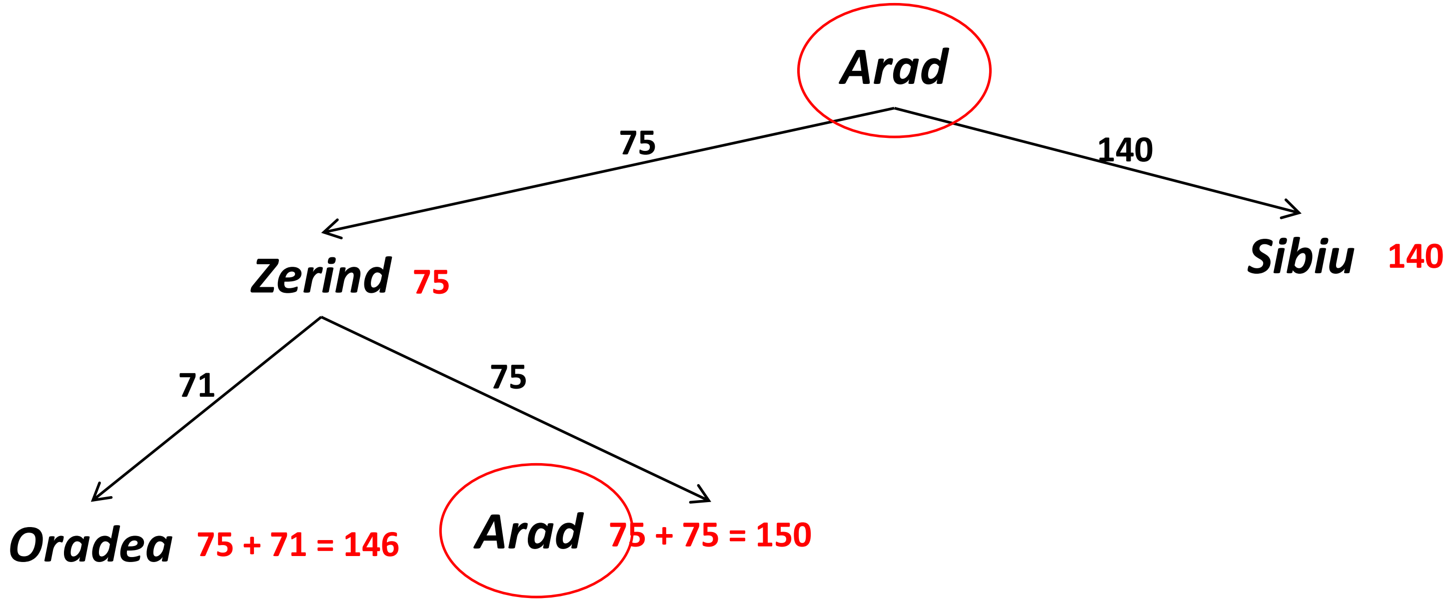
whereby n is a particular node



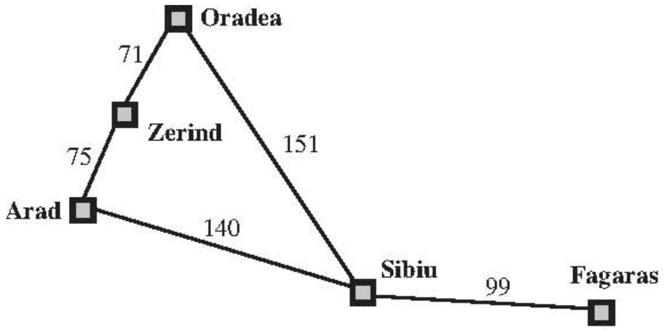


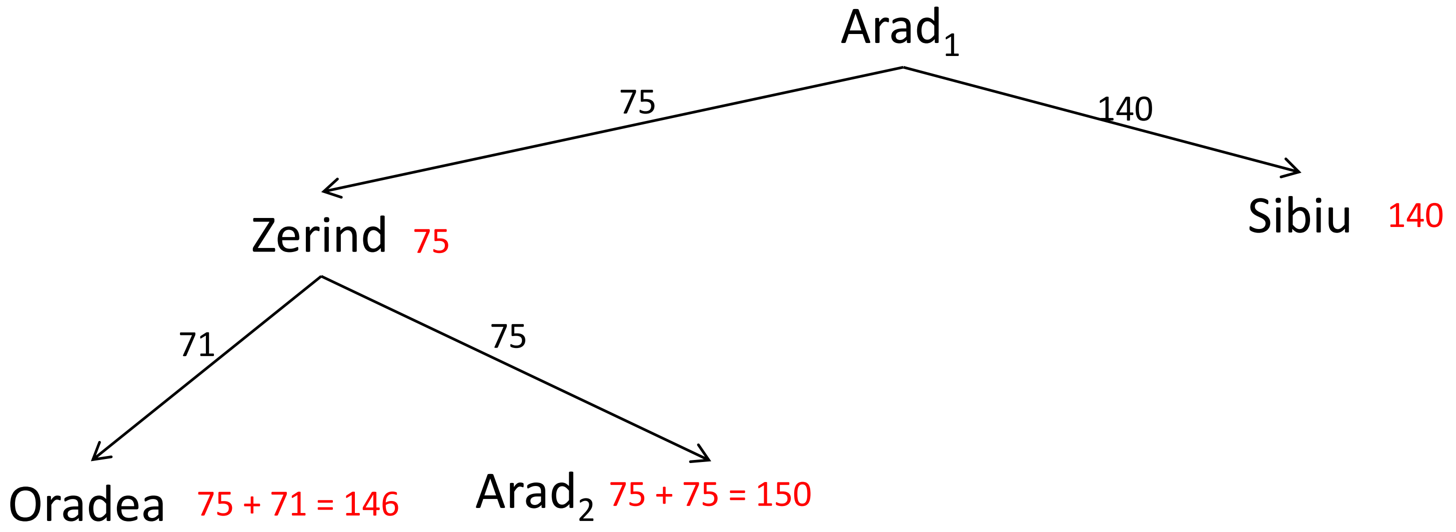
Cost function g(n)



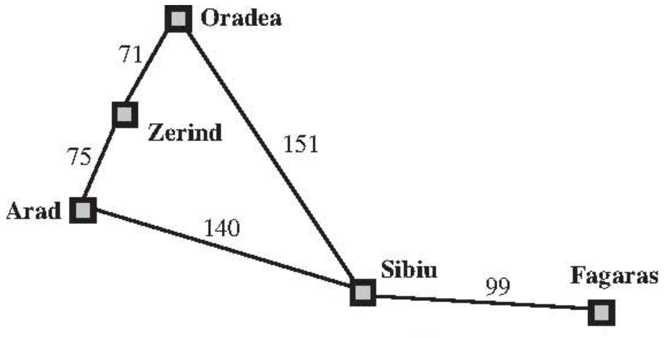


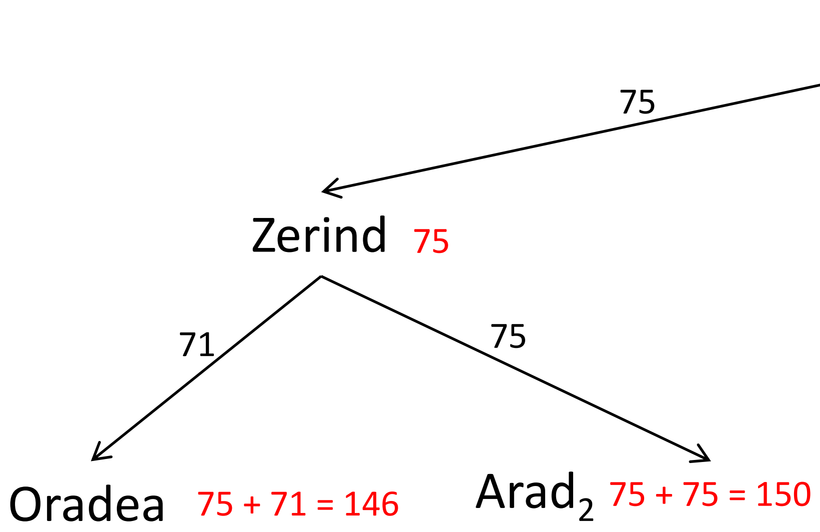
Cost function g(n)





Cost function g(n)





$g(n = Arad_1) = 0$

$g(n = Zerind) = 75$

$g(n = Sibiu) = 140$

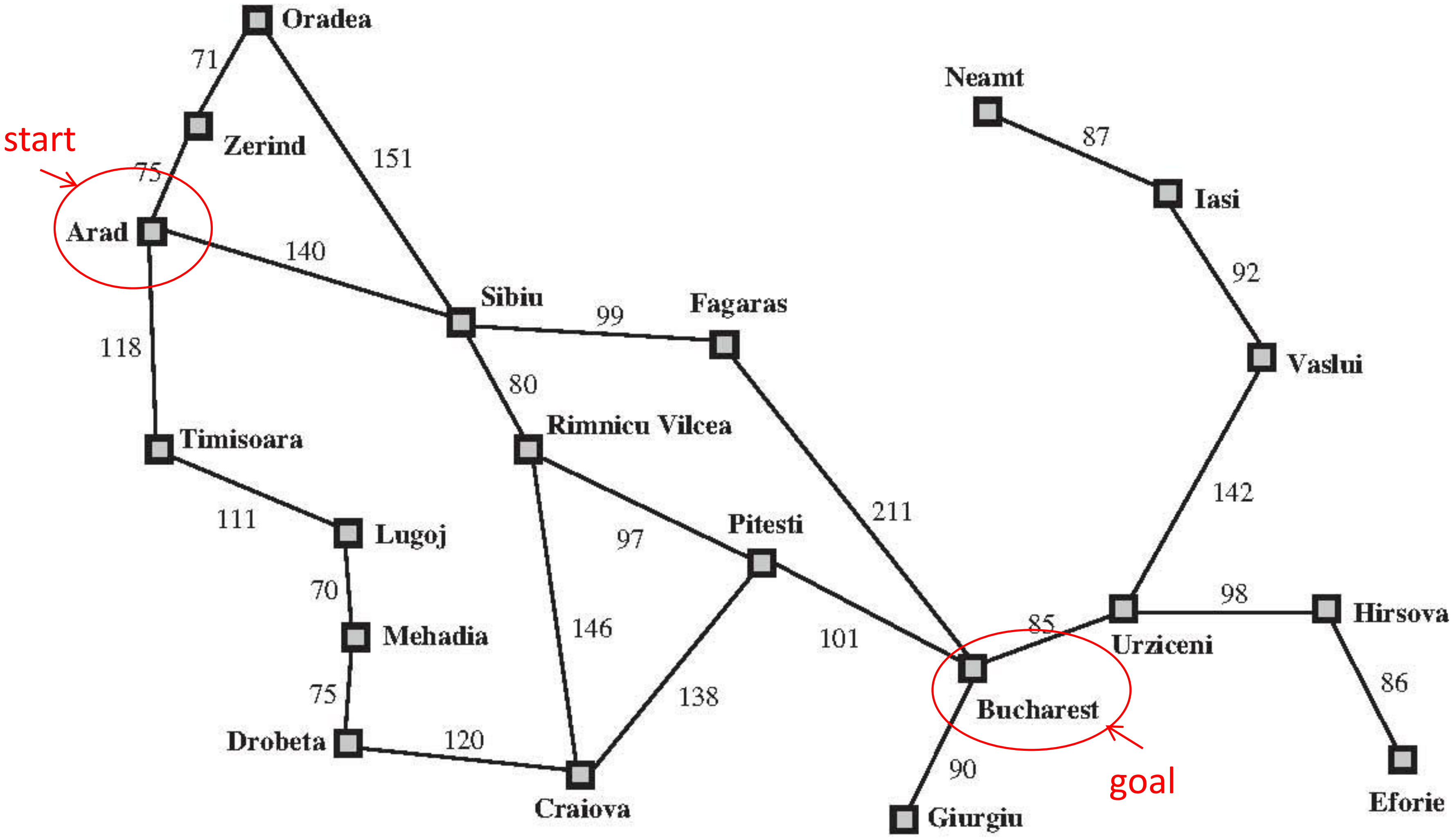
$g(n = Oradea) = 146$

$g(n = Arad_2) = 150$

Informed Search

Heuristic Search

Map of Rumania



Heuristic function h(n)

Straight Line Distances from cities in Rumania to Bucharest

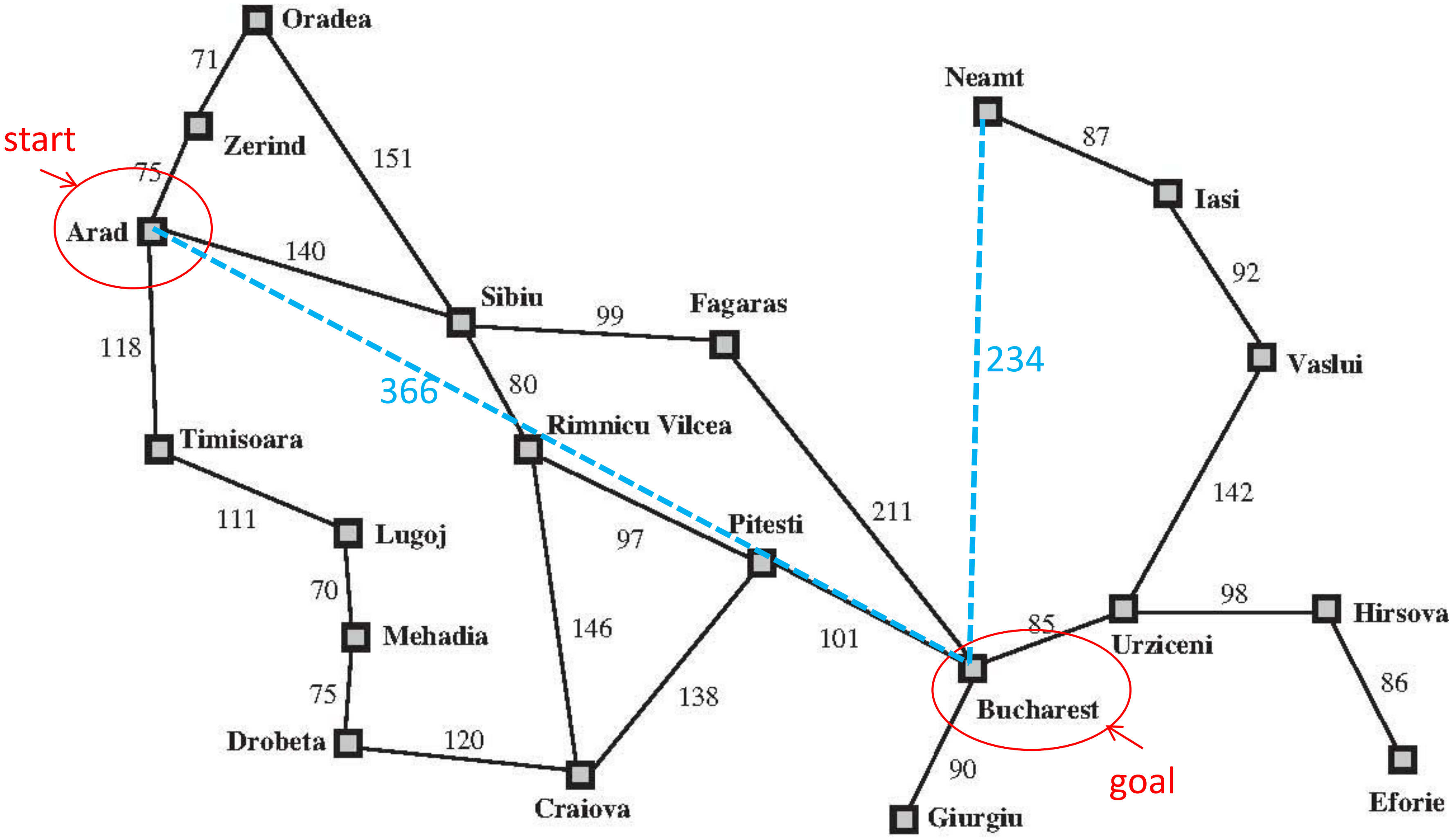
Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Drobeta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374

h(Arad) = 366

h(Bucharest) = 0

h(Oradea) = 380

Map of Rumania



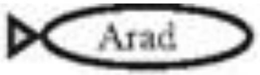
Admissibility

A heuristic function $h(n)$ is admissible if it never overestimates the cost to the goal.

Greedy best first search

Expands the nodes where h(n) is the lowest

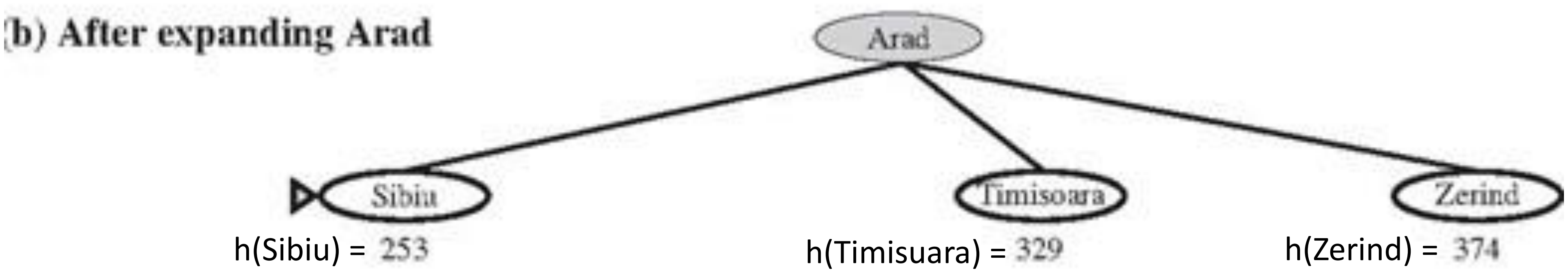
(a) The initial state



h(n) = h(Arad) = 366

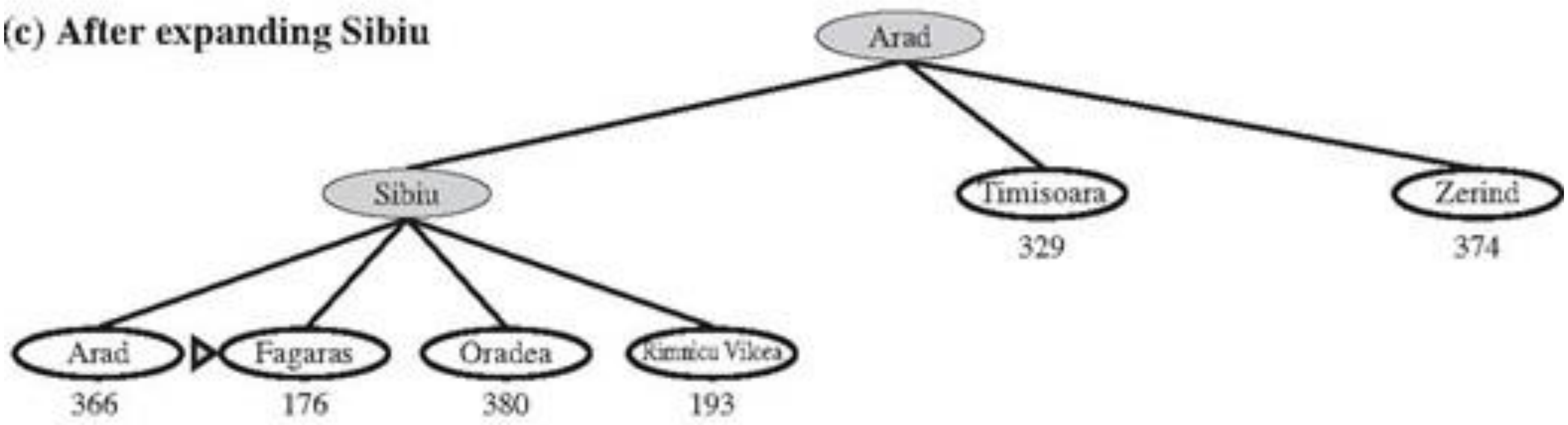
Greedy best first search

Expands the nodes where h(n) is the lowest



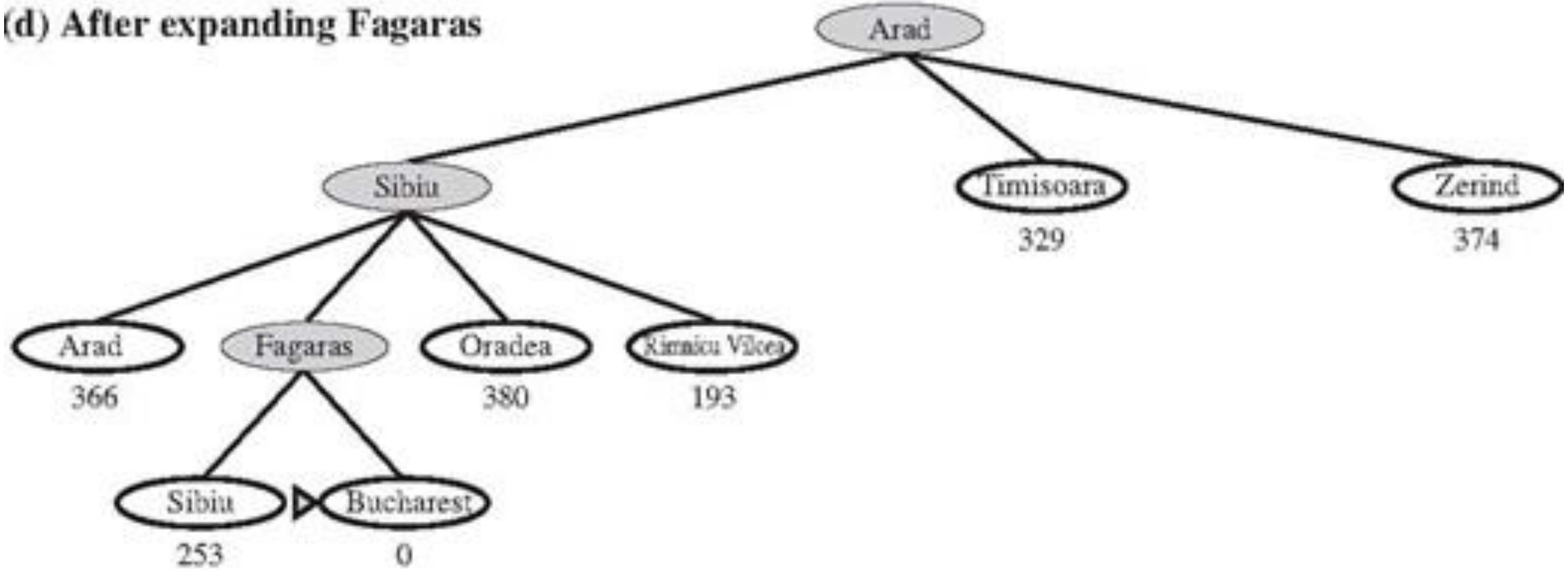
Greedy best first search

Expands the nodes where h(n) is the lowest



Greedy best first search

Expands the nodes where h(n) is the lowest



Result

Greedy best first search:

Arad -> Sibiu -> Fagaras -> Bucharest

g(Bucharest) = 140 + 99 + 211 = 450

The cheapest way:

Arad -> Sibiu -> Rimnicu Vilcea -> Pitesti -> Bucharest

g(Bucharest) = 140 + 80 + 97 + 101 = 418

Evaluation function

$$f(n) = g(n) + h(n)$$

Evaluation function

cost function

heuristic function

A* search

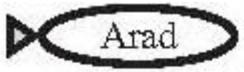
Allways expand the node where

$$\mathbf{f(n) = g(n) + h(n)}$$

is lowest

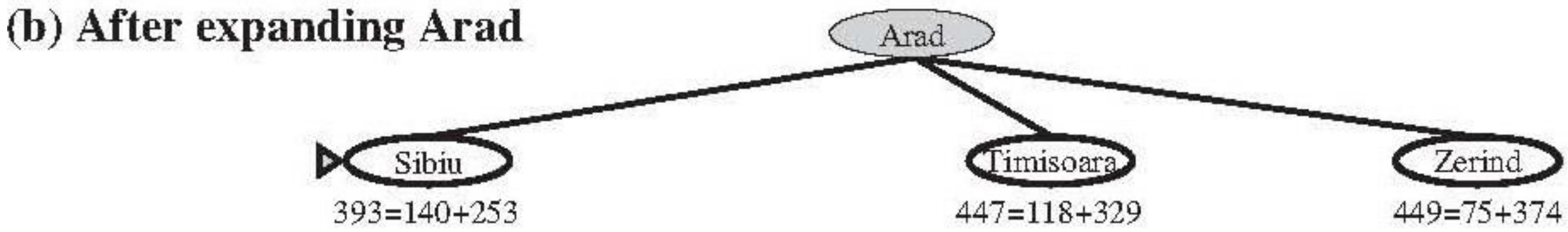
A* search

(a) The initial state



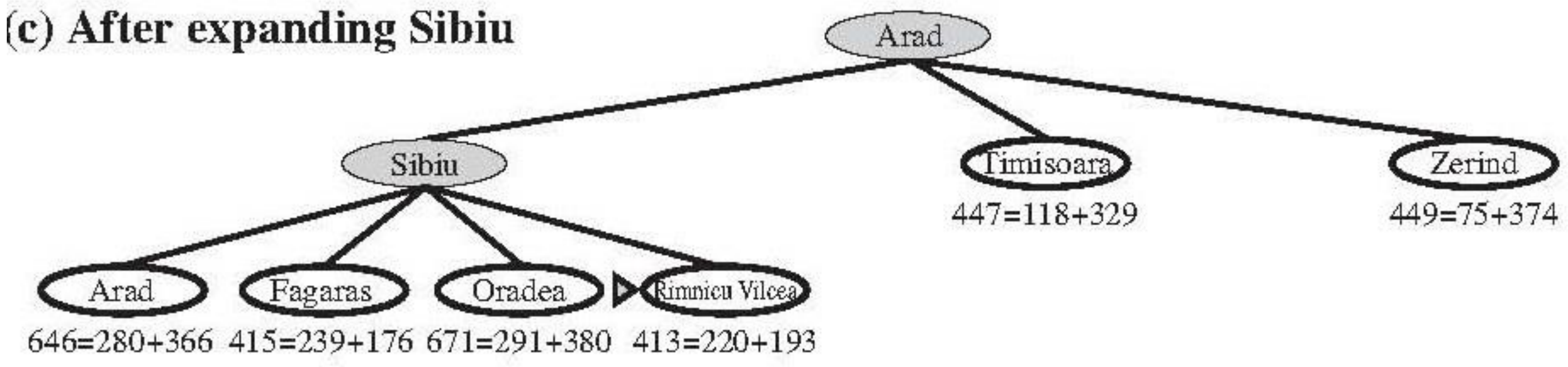
f(Arad) = g(Arad) + h(Arad) = 366=0+366

A* search

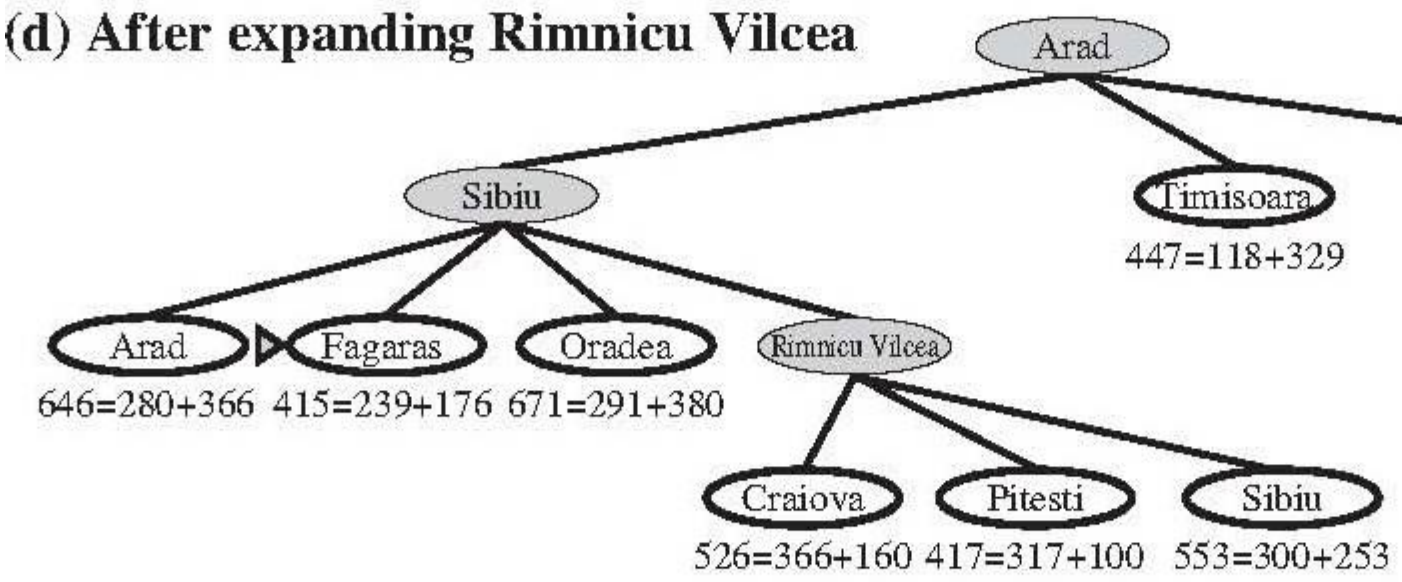


(b) After expanding Arad
Arad
Sibiu
393=140+253
Timisoara
447=118+329
Zerind
449=75+374

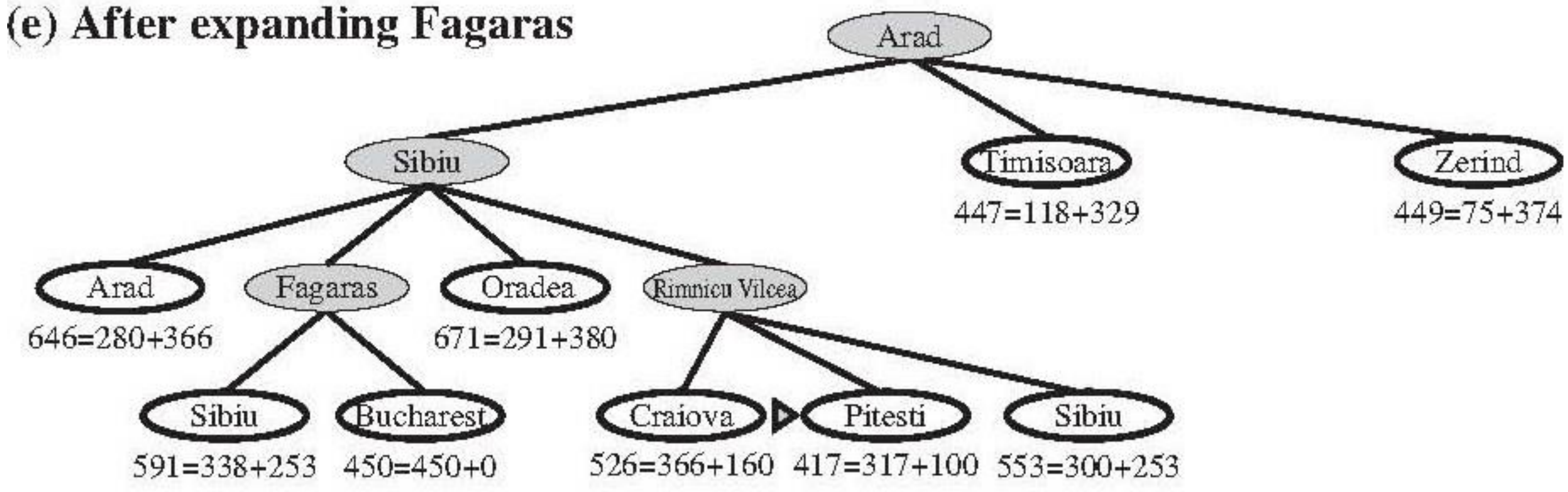
A* search


(c) After expanding Sibiu
Arad
Sibiu
Timisoara
447=118+329
Zerind
449=75+374
Arad
646=280+366
Fagaras
415=239+176
Oradea
671=291+380
Rimnicu Vilcea
413=220+193

A* search

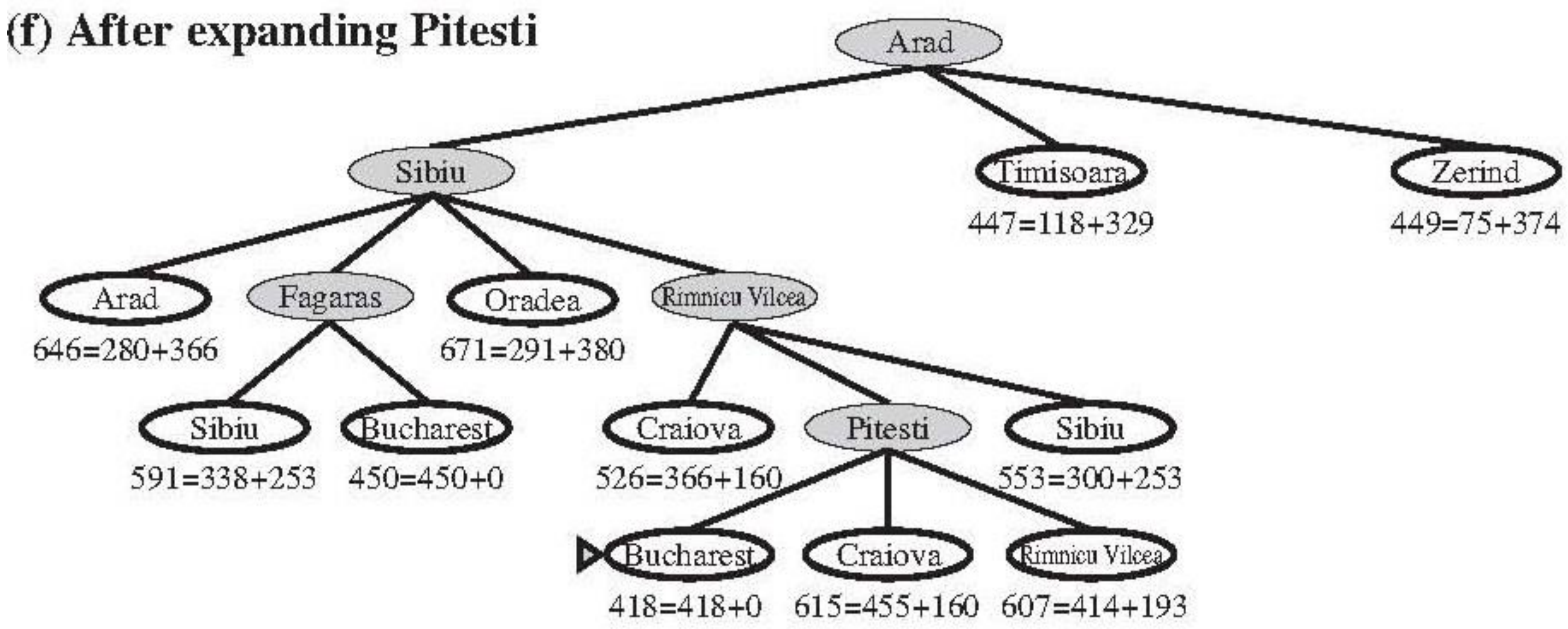

(d) After expanding Rimnicu Vilcea
Arad
Sibiu
Timisoara
447=118+329
Zerind
449=75+374
Arad
646=280+366
Fagaras
415=239+176
Oradea
671=291+380
Rimnicu Vilcea
Craiova
526=366+160
Pitesti
417=317+100
Sibiu
553=300+253

A* search



(e) After expanding Fagaras
Arad
Sibiu
Timisoara
447=118+329
Zerind
449=75+374
Arad
646=280+366
Fagaras
Oradea
671=291+380
Rimnicu Vilcea
Sibiu
591=338+253
Bucharest
450=450+0
Craiova
526=366+160
Pitesti
417=317+100
Sibiu
553=300+253

A* search


(f) After expanding Pitesti
Arad
Sibiu
Timisoara
447=118+329
Zerind
449=75+374
Arad
646=280+366
Fagaras
Oradea
671=291+380
Rimnicu Vilcea
Sibiu
591=338+253
Bucharest
450=450+0
Craiova
526=366+160
Pitesti
Sibiu
553=300+253
Bucharest
418=418+0
Craiova
615=455+160
Rimnicu Vilcea
607=414+193

Performance of search algorithms

- b : branching factor
- d: depth of the shallowest solution
- m: maximum depth in the search tree
- l: depth limit

Measure:

- How long will it take to find a solution?
- How much memory is needed?
- Is it complete? (Does it find a solution if there is one?)
- Is it optimal? (Does it find the best/shallowest/cheapest solution?)

Evaluation of A* search

- Optimal
  when the heuristic is admissible

- Complete
  under the condition that there are only finitely many nodes with less or equal cost as the optimal solution.

Literature

Artificial Intelligence

A modern approach

Third Edition

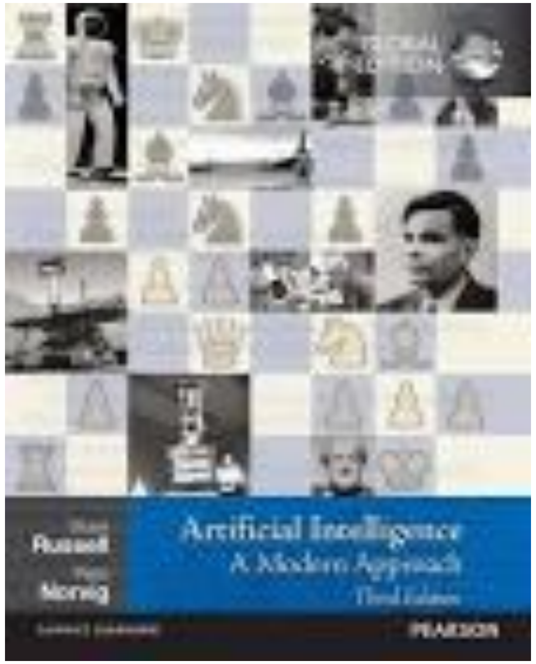
Stuart Russel & Peter Norvig

2010 by Pearson Education Inc.

ISBN-13 978-0-13-207148-2

IISBN-10 0-13-207148-7

<u>Chapters: 3 and 4</u>



Advanced Artificial Intelligence

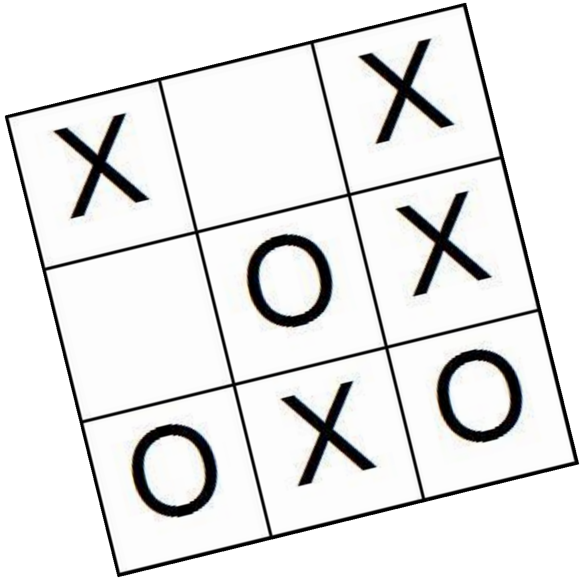
Game Theory

Game Theory

Adversarial Search

Game Theory/Adversarial Search

- Multi agent system
- Competitive environment
  - E.g. in games
    - one agent wants to win over an opponent



Characteristics of games

- Chess
  - Branching factor: about 35
  - Moves per player: about 50
  - Search tree: about 35^{100} nodes
- Backgammon
  - Includes the element of chance
- Bridges
  - Includes imperfect information
- Multi player games
  - With more than two opponents

Search in Games

- It has to be good
  - (otherwise you get bored or you won’t sell the program)
- It has to be fast (time matters)

Formal definition of a game

s = state, p = player, a = action (move)

• s_0	Initial state
• Player(s)	which move it is in state s
• Action(s)	legal moves in state s
• Result(s, a)	result of move
• Terminal test (s)	true when game is over (terminal state)
• Utility(s, p)	e.g. win, loss, draw (1, 0, 0.5)

Zero-sum games

- Games where the sum of the outcome is the same for every instance of the game

- E.g. chess, tic-tac-toe
  - One winner, one looser (1 + 0 = 1)
  - Draw (0.5 + 0.5 = 1)

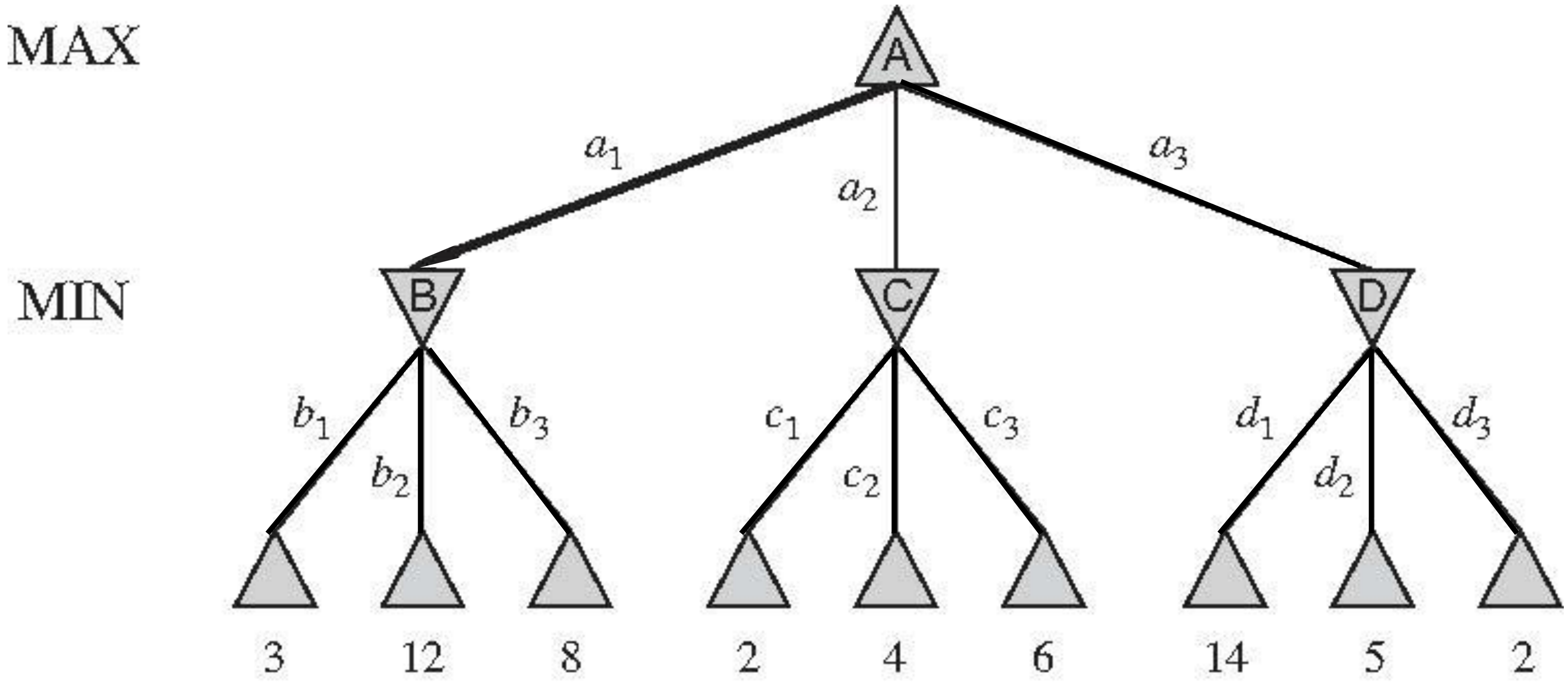
“Minimax”

- You want to win
  - maximizing the outcome for you
- Your opponent wants you to loose
  - Minimizing the outcome for you
- You think one move at a time
  - Your move (maximize), opponents move (minimize), your move (maximize), opponents move (minimize), your move (maximize), …
- What you do, needs to depend on what you believe will be your opponents reaction

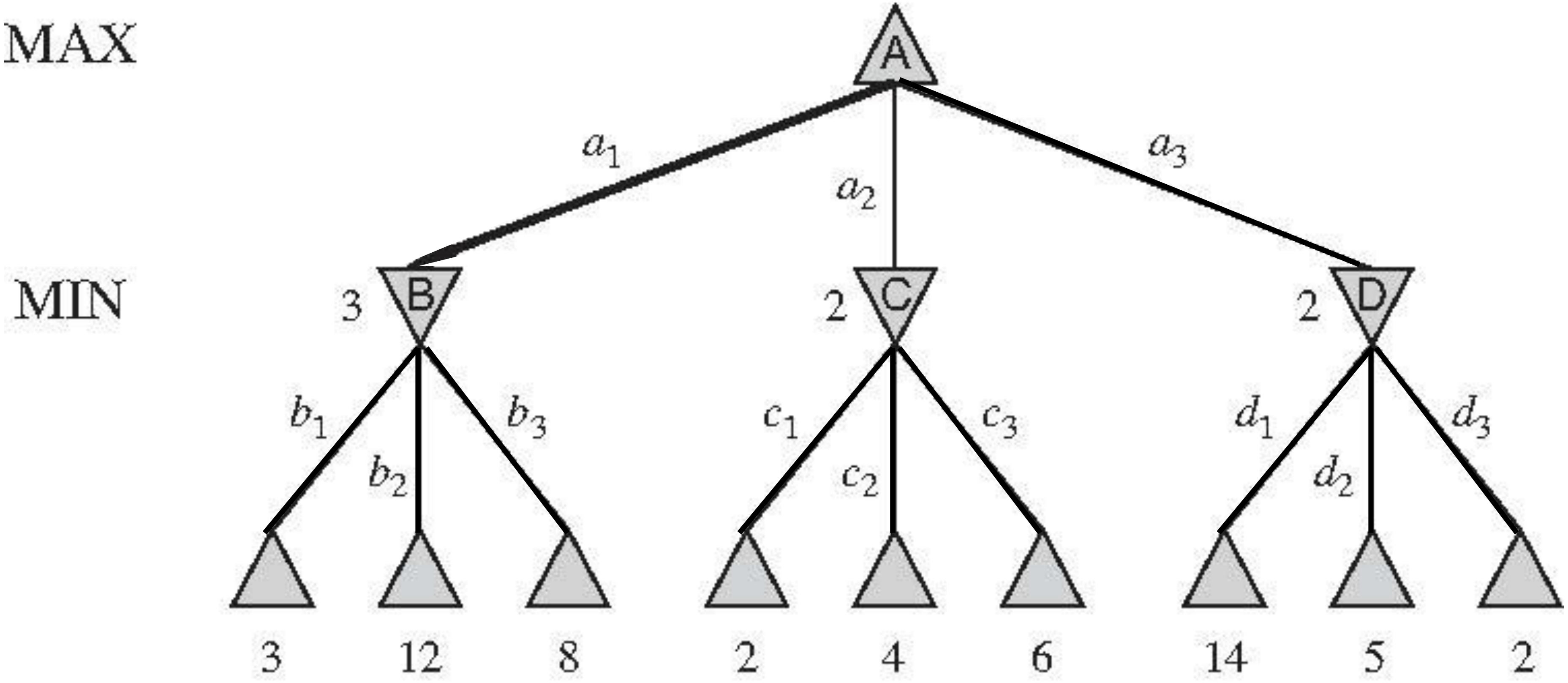
From the perspective of the game/algorithm

- The first player is always Max
  - Max wants the result to be +1
- The second player is always Min
  - Min wants the result to be -1
    (which means -1 for Max,
    which automatically is +1 for Min)

Minimax algorithm



Minimax algorithm



Minimax algorithm