Lecture 3: Uninformed search

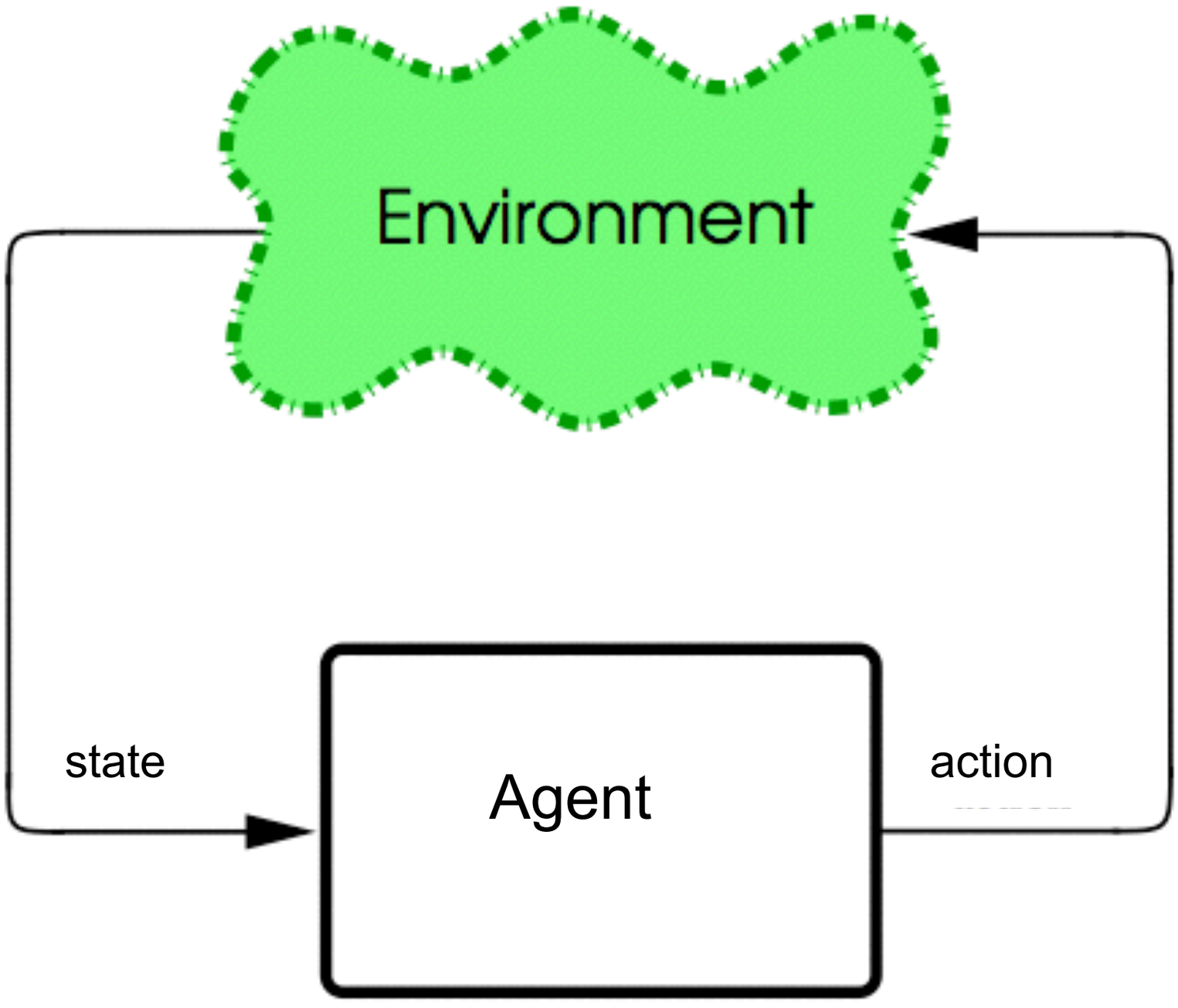
Artificial Intelligence
CS-UY-4613-A / CS-GY-6613-I

Julian Togelius
julian.togelius@nyu.edu

On the menu

- Problem solving as search
- Uninformed search algorithms
- Benchmark problems
- The project

An agent



Problem-solving as search

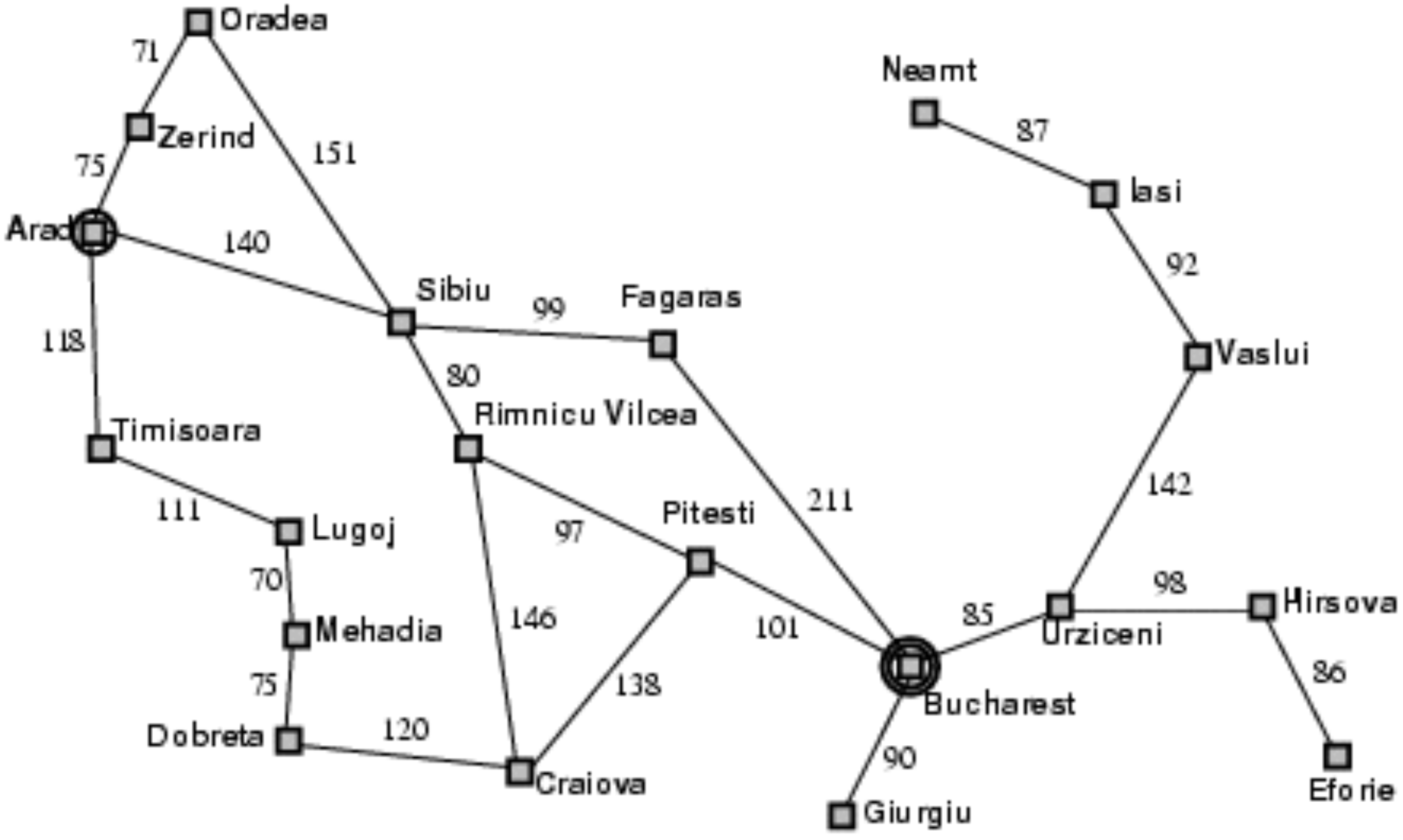
- An agent is in a *state*
- It wants to get to a *goal state*
- It needs to find the sequence of actions that gets it there
  - (cheapest, fastest, safest…)

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 do
        goal ← FORMULATE-GOAL(state)
        problem ← FORMULATE-PROBLEM(state, goal)
        seq ← SEARCH(problem)
action ← FIRST(seq)
seq ← REST(seq)
return action
```

From Arad to Bucharest



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
    - Agent may have no idea where it is; solution is a sequence
- Nondeterministic and/or partially observable: contingency problem
    - percepts provide new information about current state
- Unknown state space: exploration problem

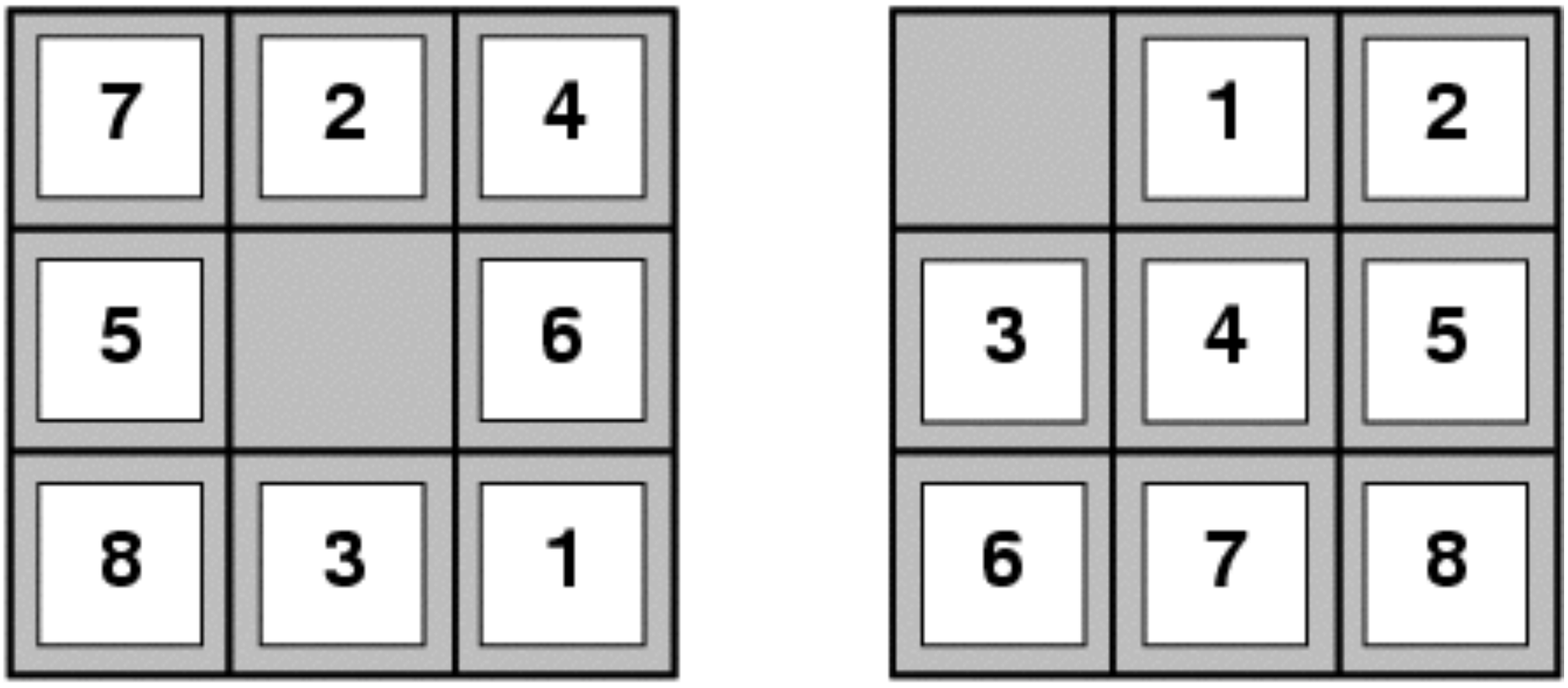
What problem type is…

- Driving to Bucharest?
- Flying to Bucharest?
- Playing Pac-Man?
- Playing StarCraft?
- Playing Poker?
- Living a good life?

Problem components

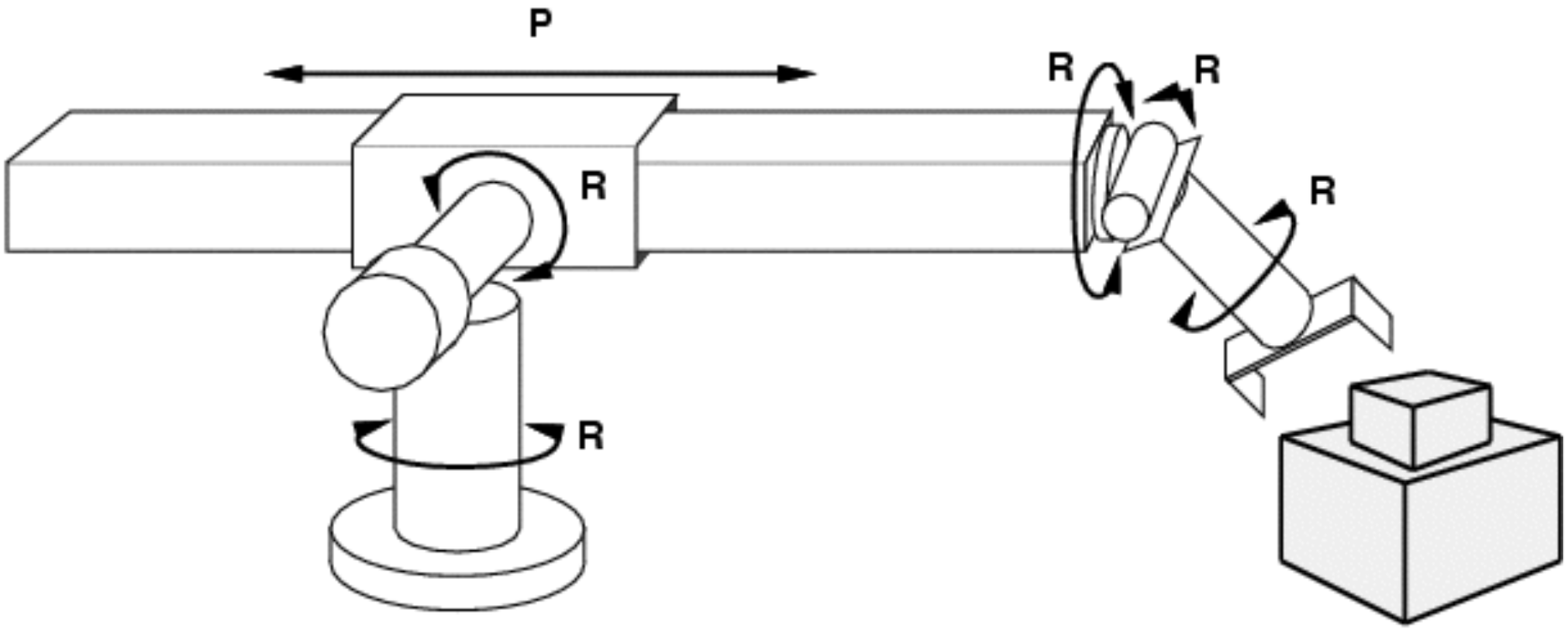
- Initial state
- Actions and successor function
  S(state, action) -> state’
- Goal test (are we there yet?)
- Path cost
- A solution is a sequence of actions leading from the initial state to a goal state

Problem components?



Start State **Goal State**

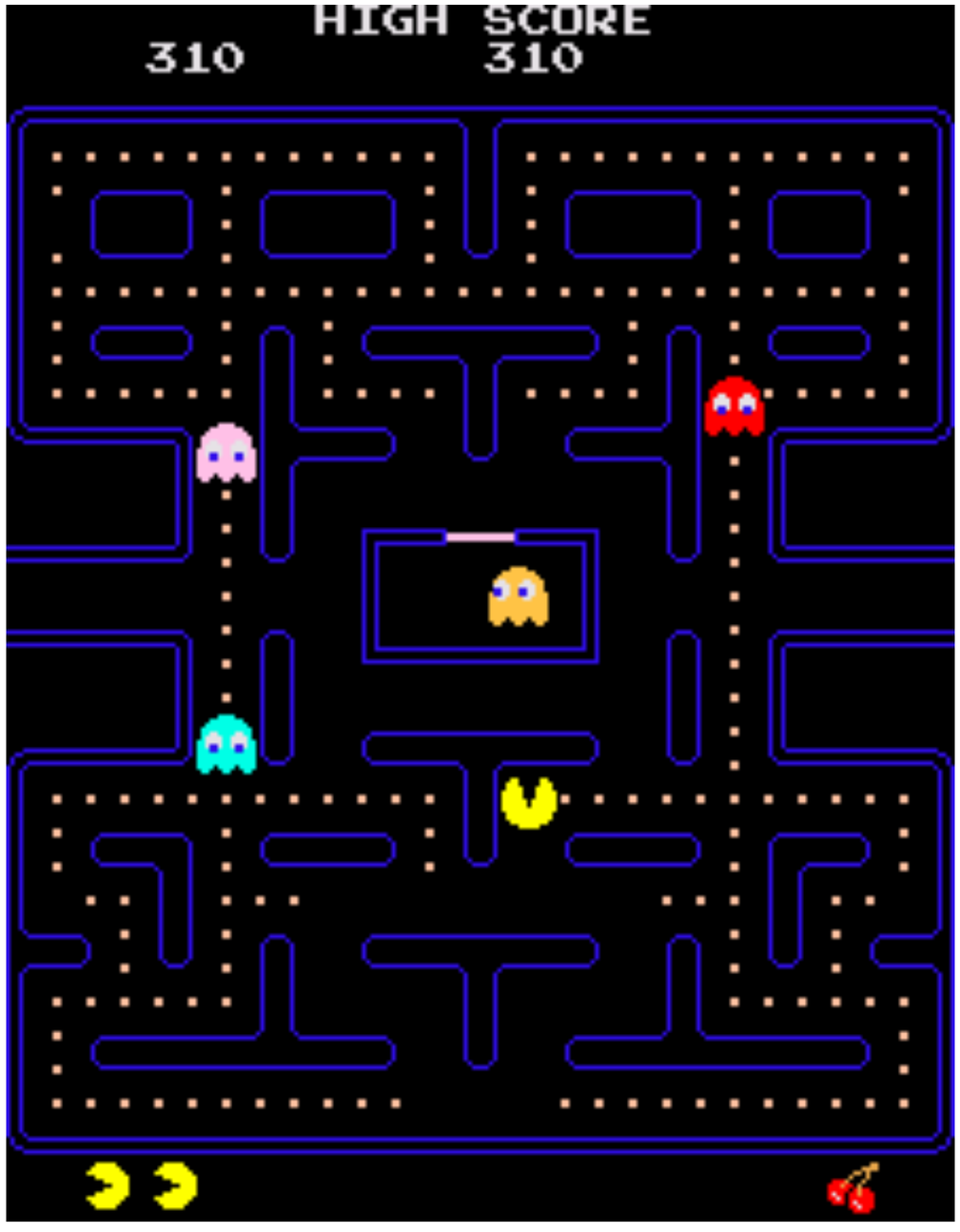
Problem components?



Problem components?

Problem components?

Problem components?



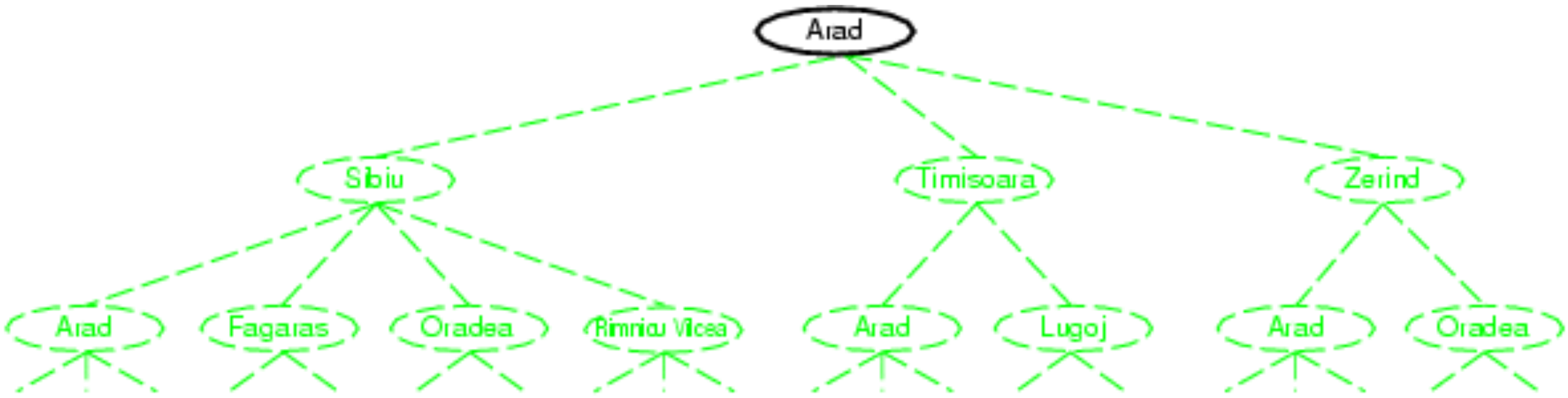
Problem components?

Tree search

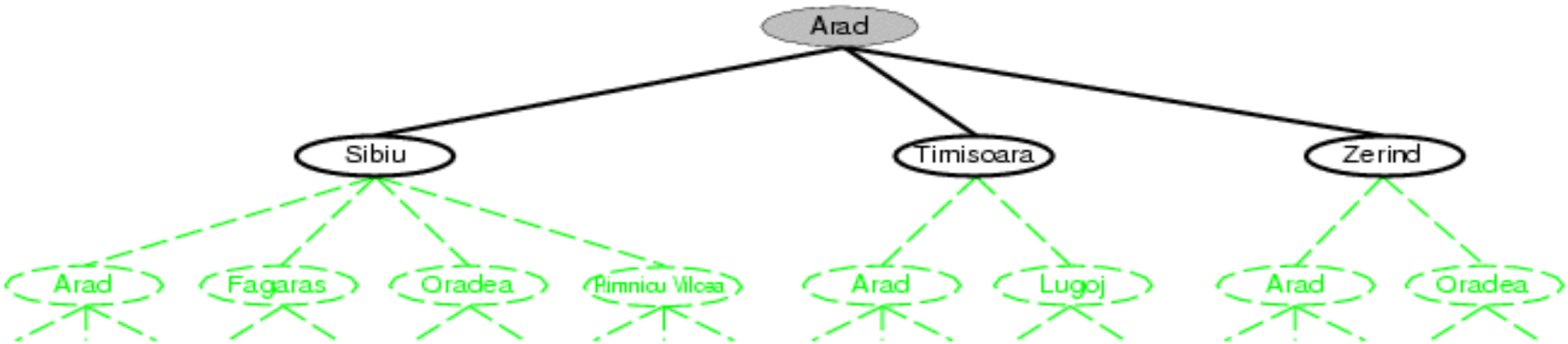
- offline, simulated exploration of state space by generating successors of already-explored states (a.k.a.~expanding states)

```
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
```

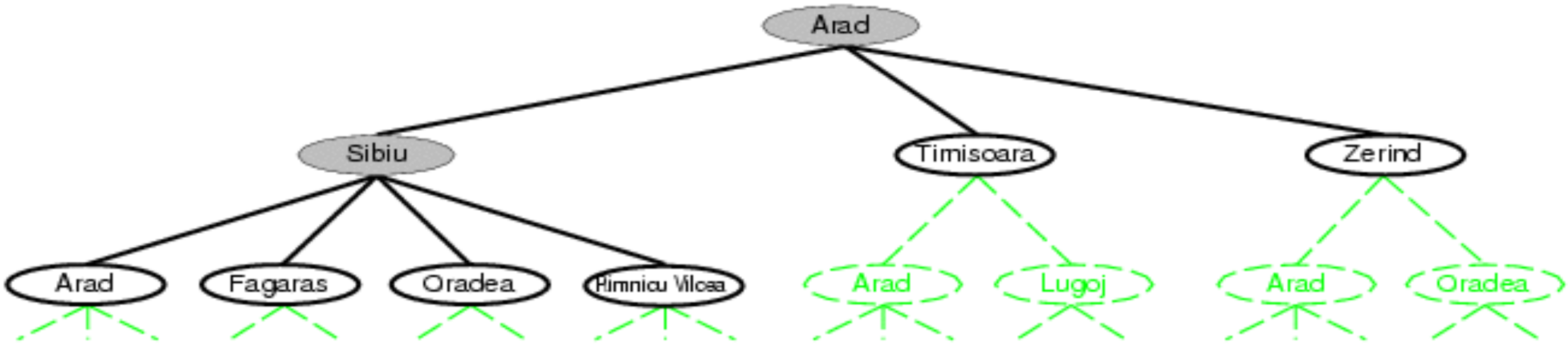
Tree search


Arad
Sibiu
Timisoara
Zerind
Arad
Fagaras
Oradea
Rimnicu Vilcea
Arad
Lugoj
Arad
Oradea

Tree search


Arad
Sibiu
Timisoara
Zerind
Arad
Fagaras
Oradea
Rimnicu Vilcea
Arad
Lugoj
Arad
Oradea

Tree search


Arad
Sibiu
Timisoara
Zerind
Arad
Fagaras
Oradea
Rimnicu Vilcea
Arad
Lugoj
Arad
Oradea

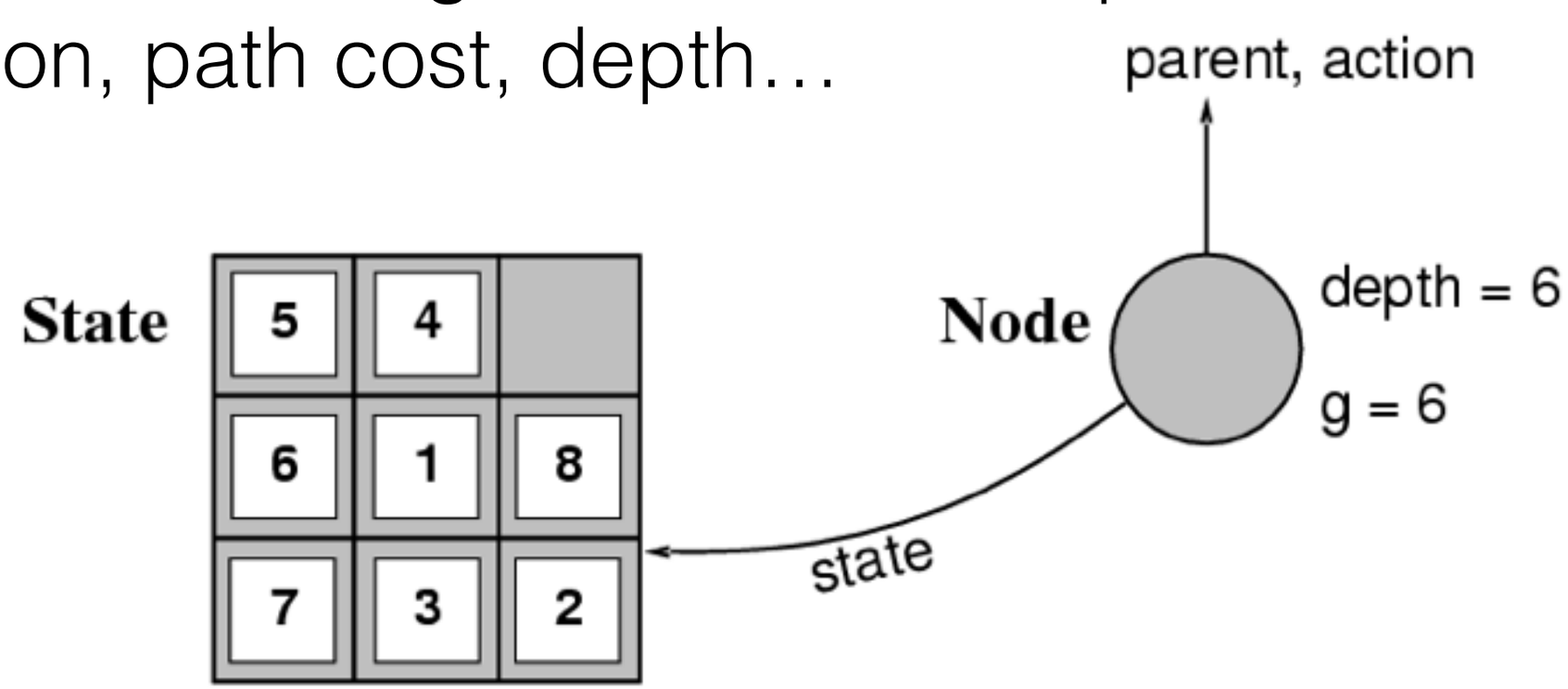
```
function TREE-SEARCH( problem, fringe) returns a solution, or failure
    fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
    loop do
        if fringe is empty then return failure
        node ← REMOVE-FRONT(fringe)
        if GOAL-TEST[problem](STATE[node]) then return SOLUTION(node)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

Nodes versus states

- A state is a configuration of an environment/agent
- A node is a data structure constituting part of a search tree, might include state, parent node, action, path cost, depth…

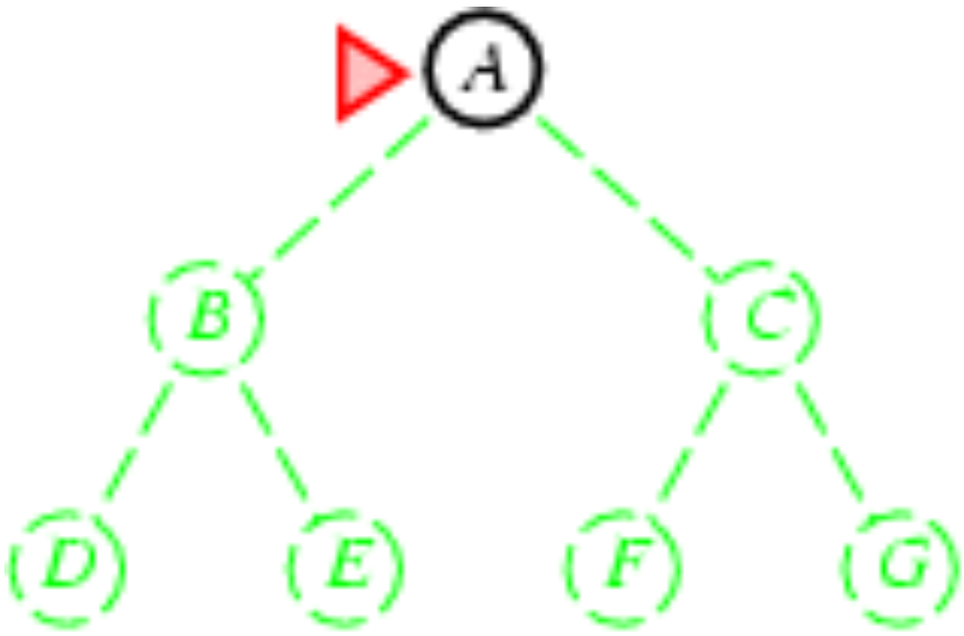


Uninformed search

- Uninformed search strategies use only the information available in the problem definition
- Breadth-first search
- Uniform cost search
- Depth-first search
- Depth-limited search
- Iterative deepening search

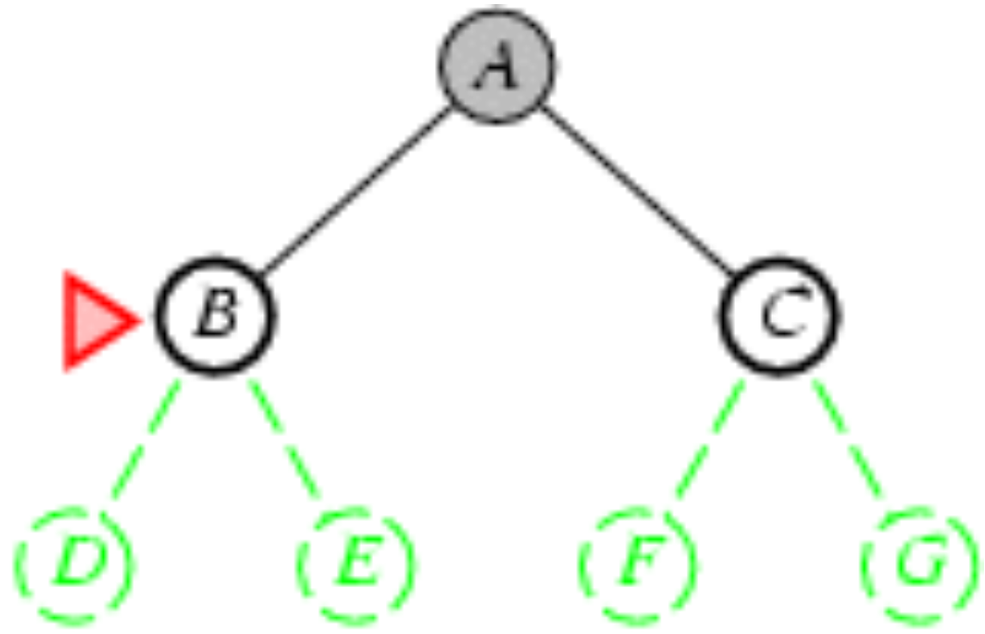
Breadth-first search

- Expand shallowest unexpanded node
- Implementation: new nodes added to a FIFO queue



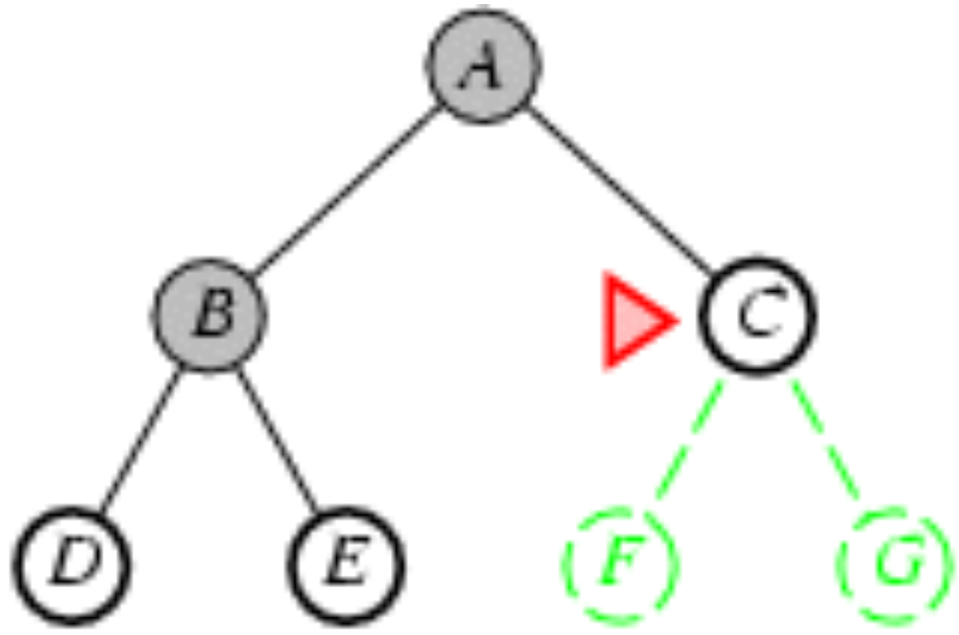
Breadth-first search

- Expand shallowest unexpanded node
- Implementation: new nodes added to a FIFO queue



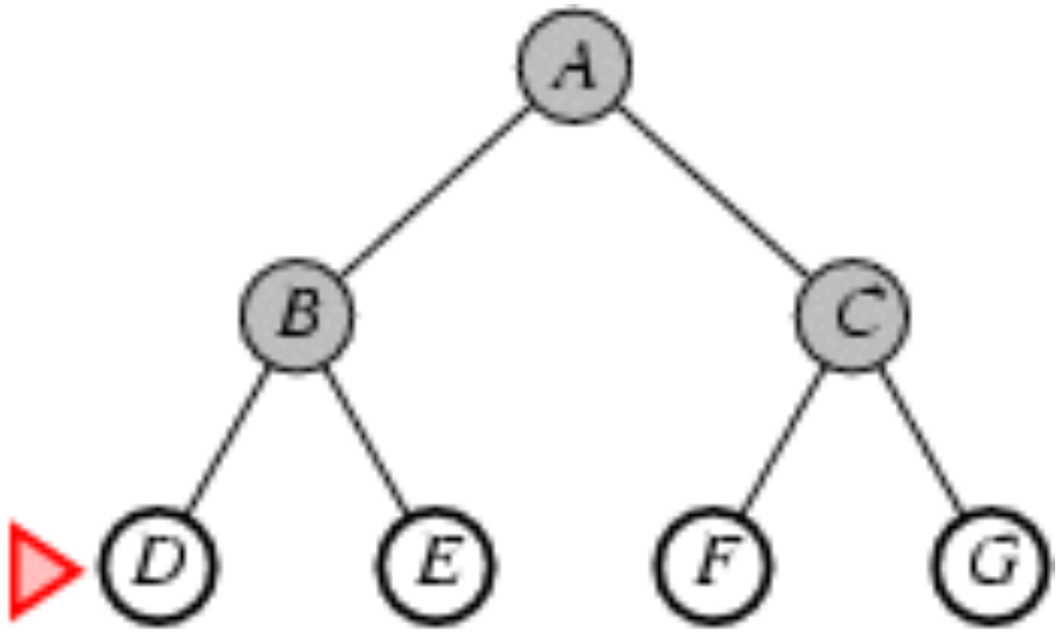
Breadth-first search

- Expand shallowest unexpanded node
- Implementation: new nodes added to a FIFO queue



Breadth-first search

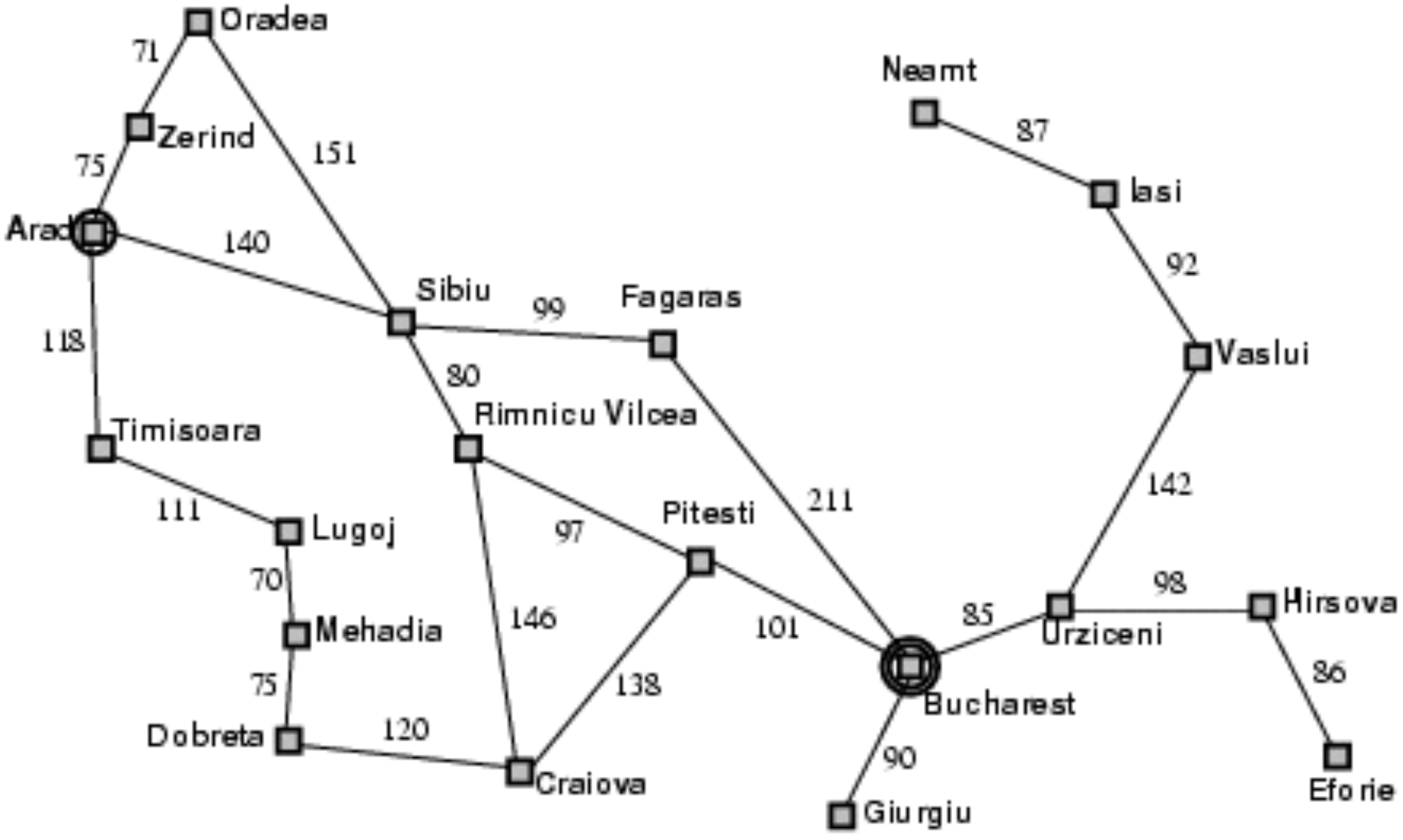
- Expand shallowest unexpanded node
- Implementation: new nodes added to a FIFO queue



Breadth-first search

- Complete? Yes (if b is finite)
- Time? $1+b+b^2+b^3+\ldots +b^d + b(b^d-1) = O(b^{d+1})$
- Space? $O(b^d+1)$ (keeps every node in memory)
- Optimal? Yes (if cost = 1 per step)

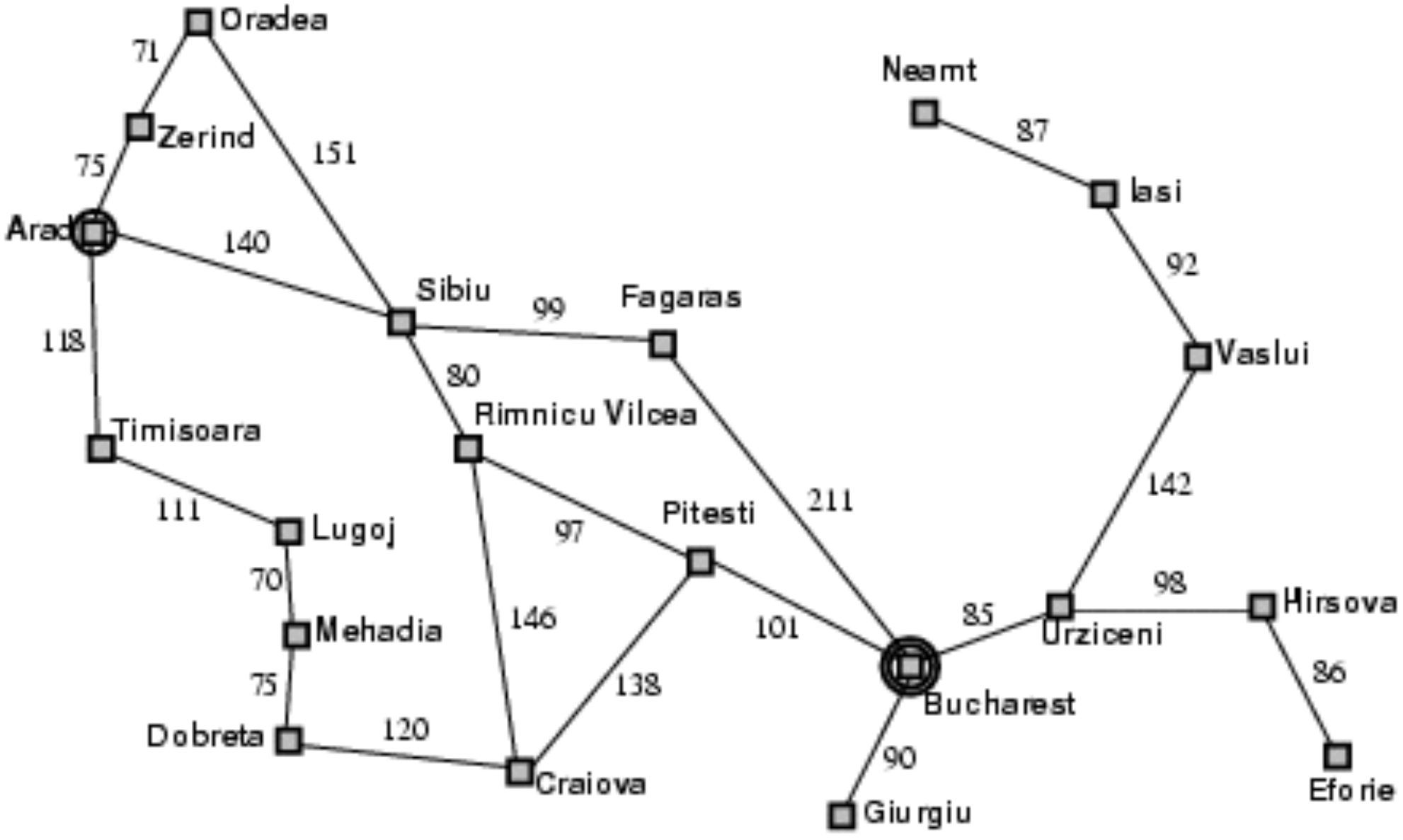
From Arad to Bucharest



Uniform-cost search

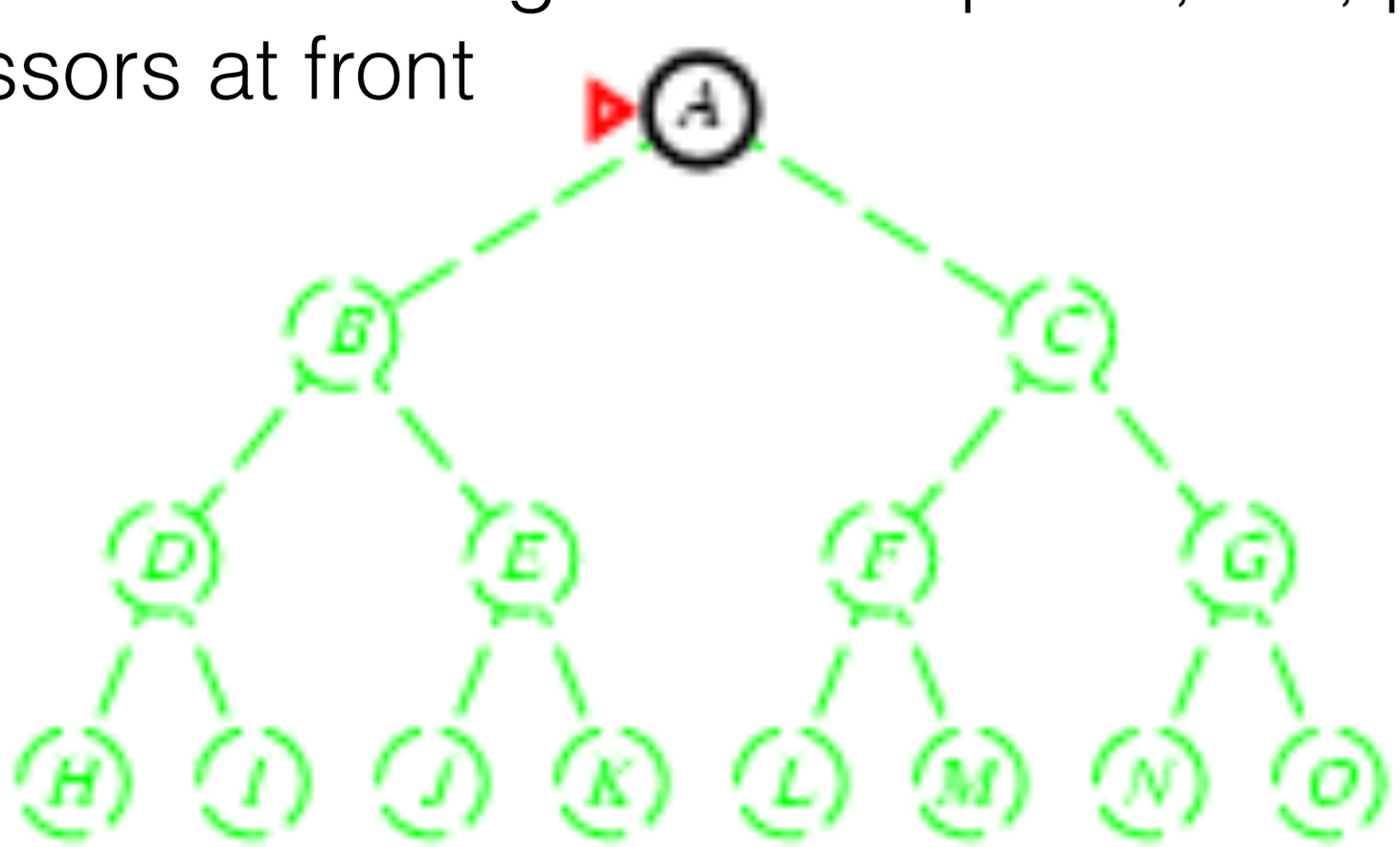
- Expand least-cost unexpanded node
- Equivalent to breadth-first if step costs all equal
- Complete and optimal

From Arad to Bucharest



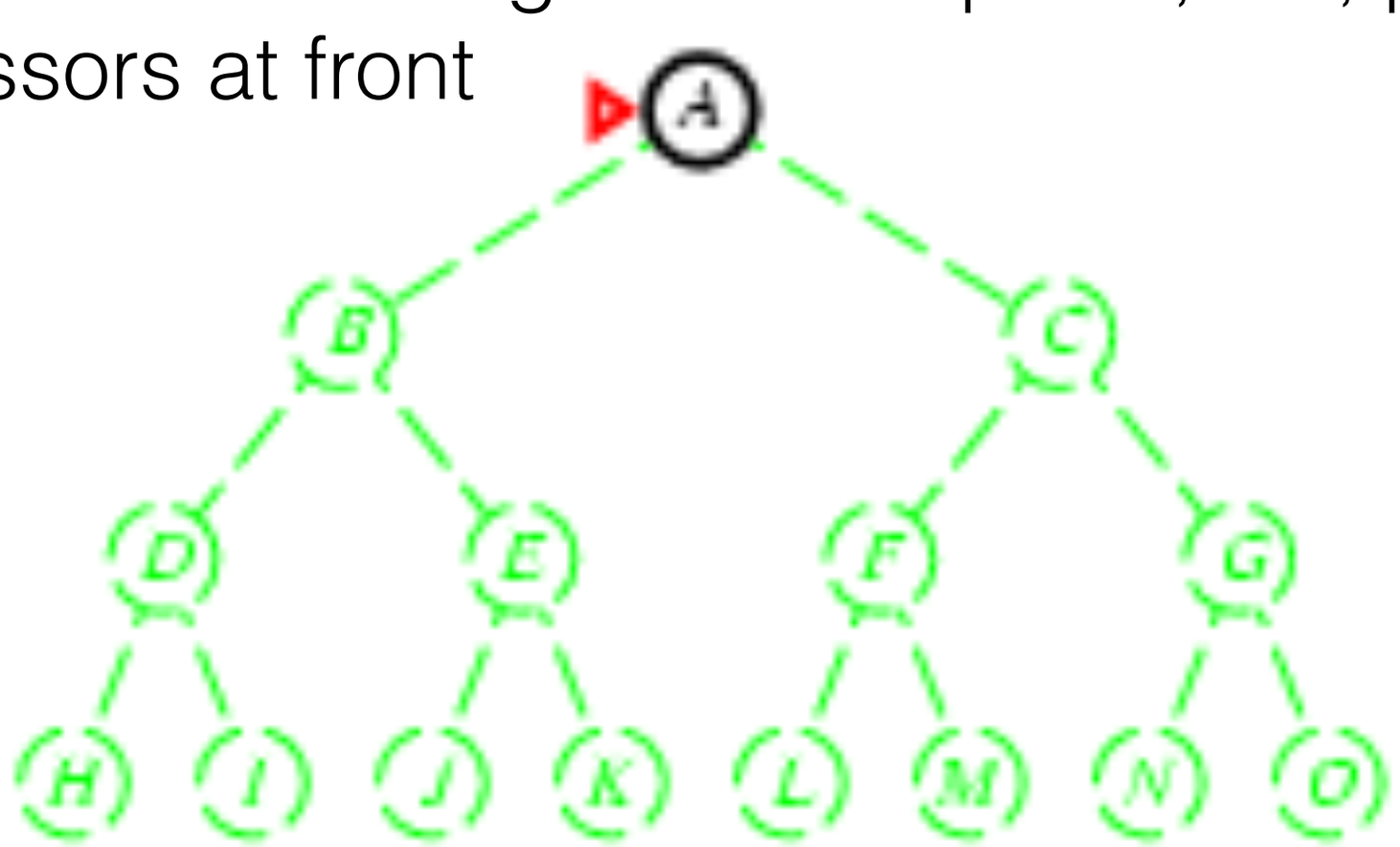
Depth-first search

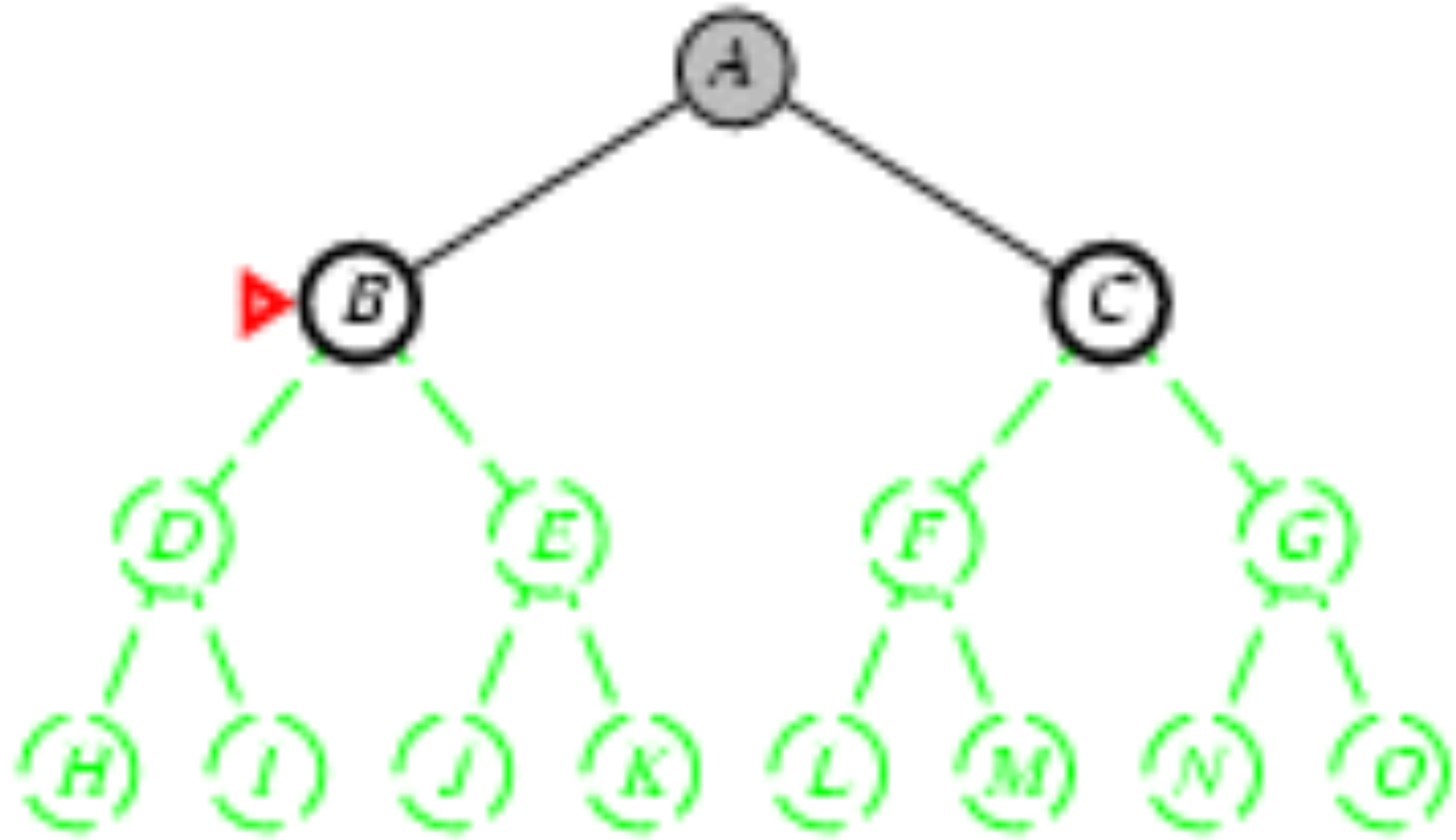
- Expand deepest unexpanded node
- Implementation: fringe = LIFO queue, i.e., put successors at front

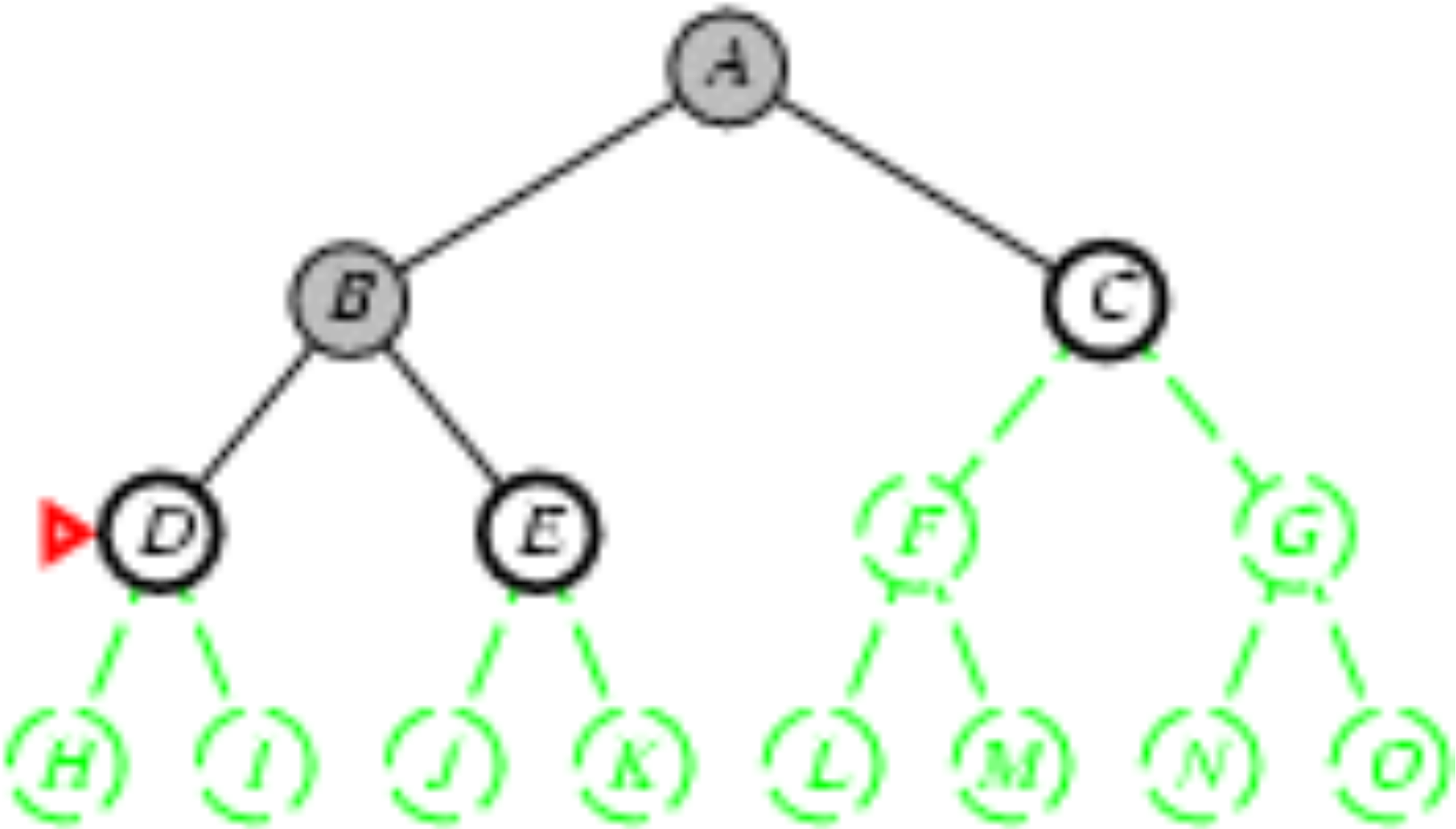


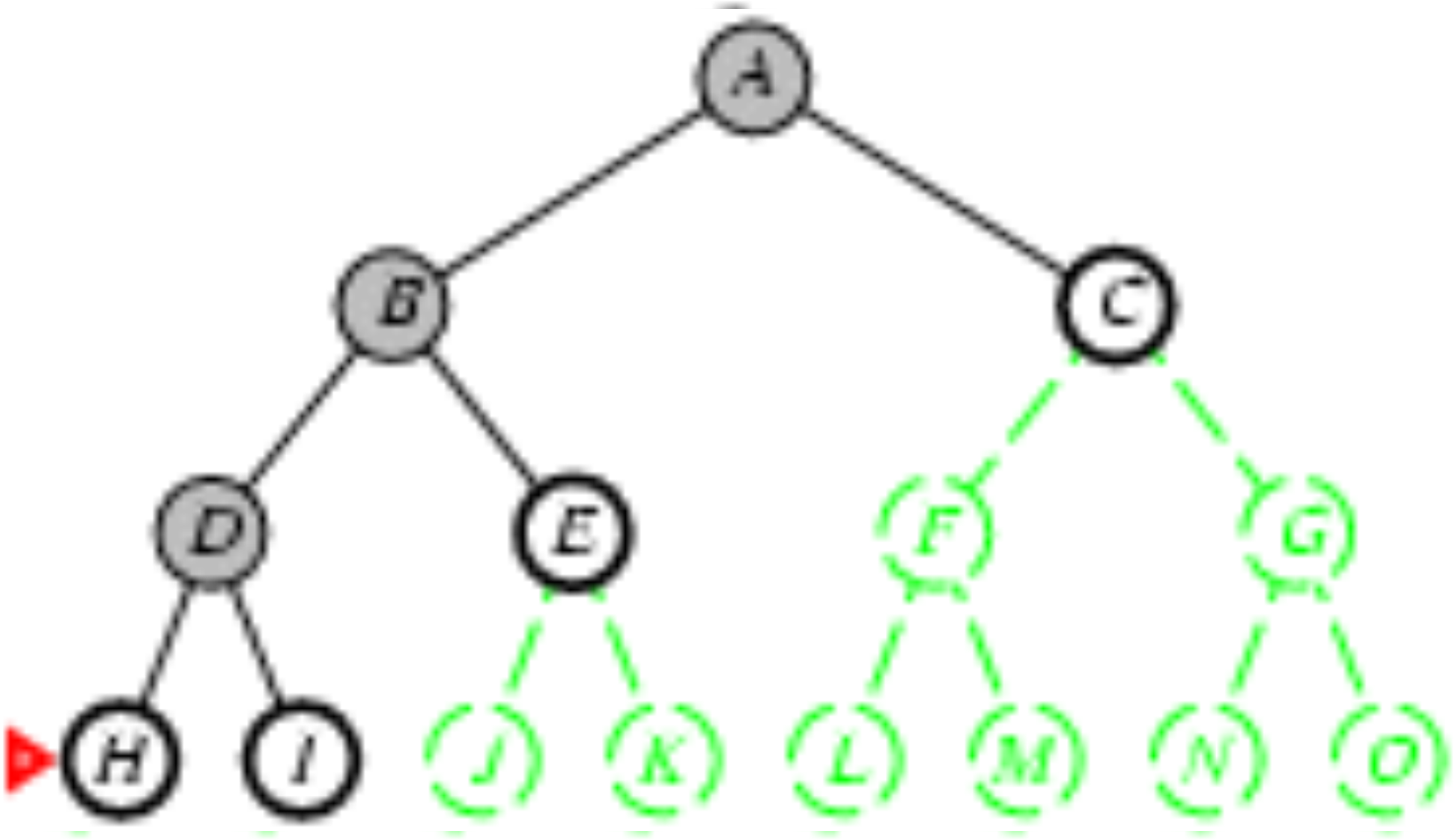
Depth-first search

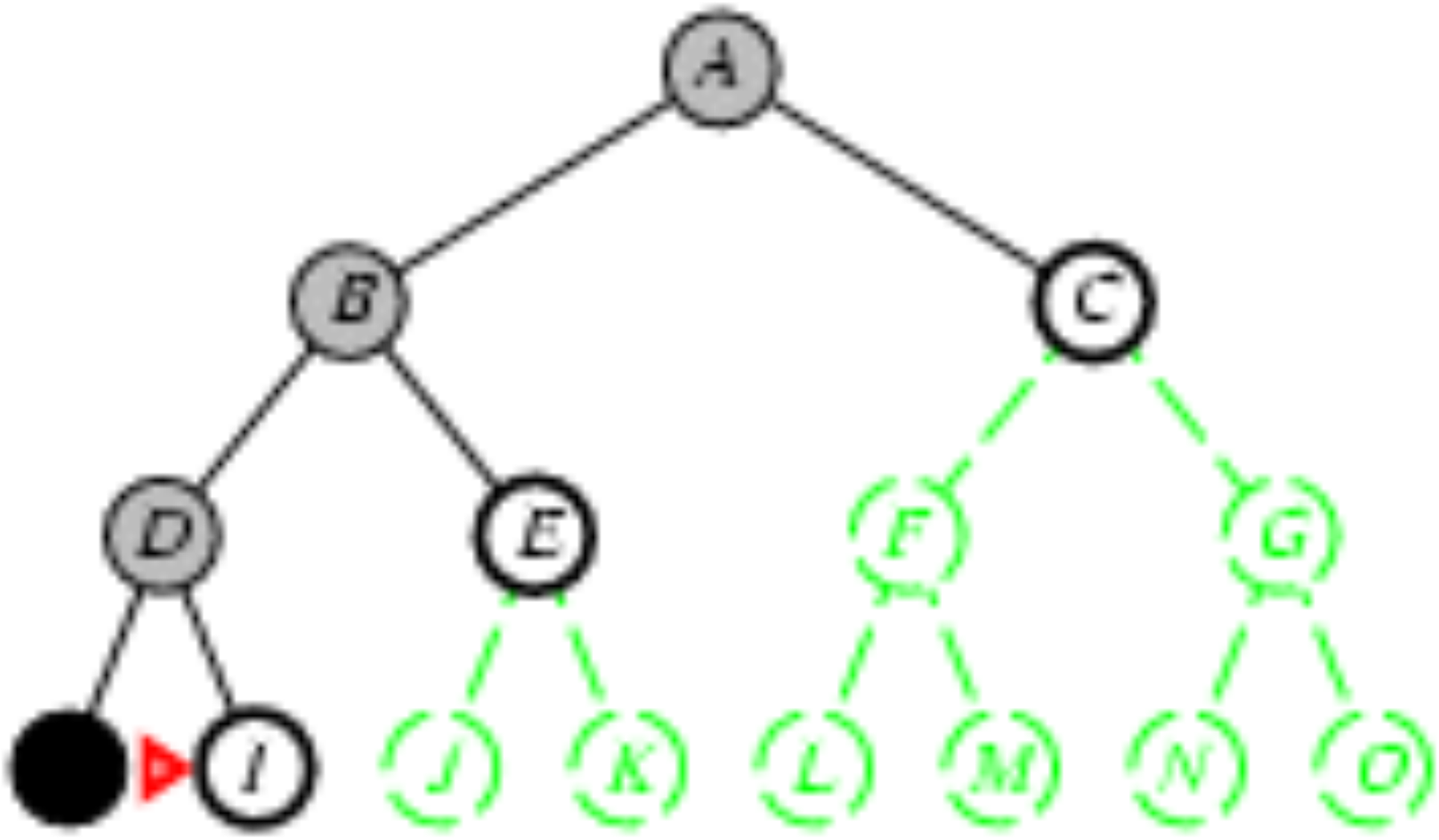
- Expand deepest unexpanded node
- Implementation: fringe = LIFO queue, i.e., put successors at front

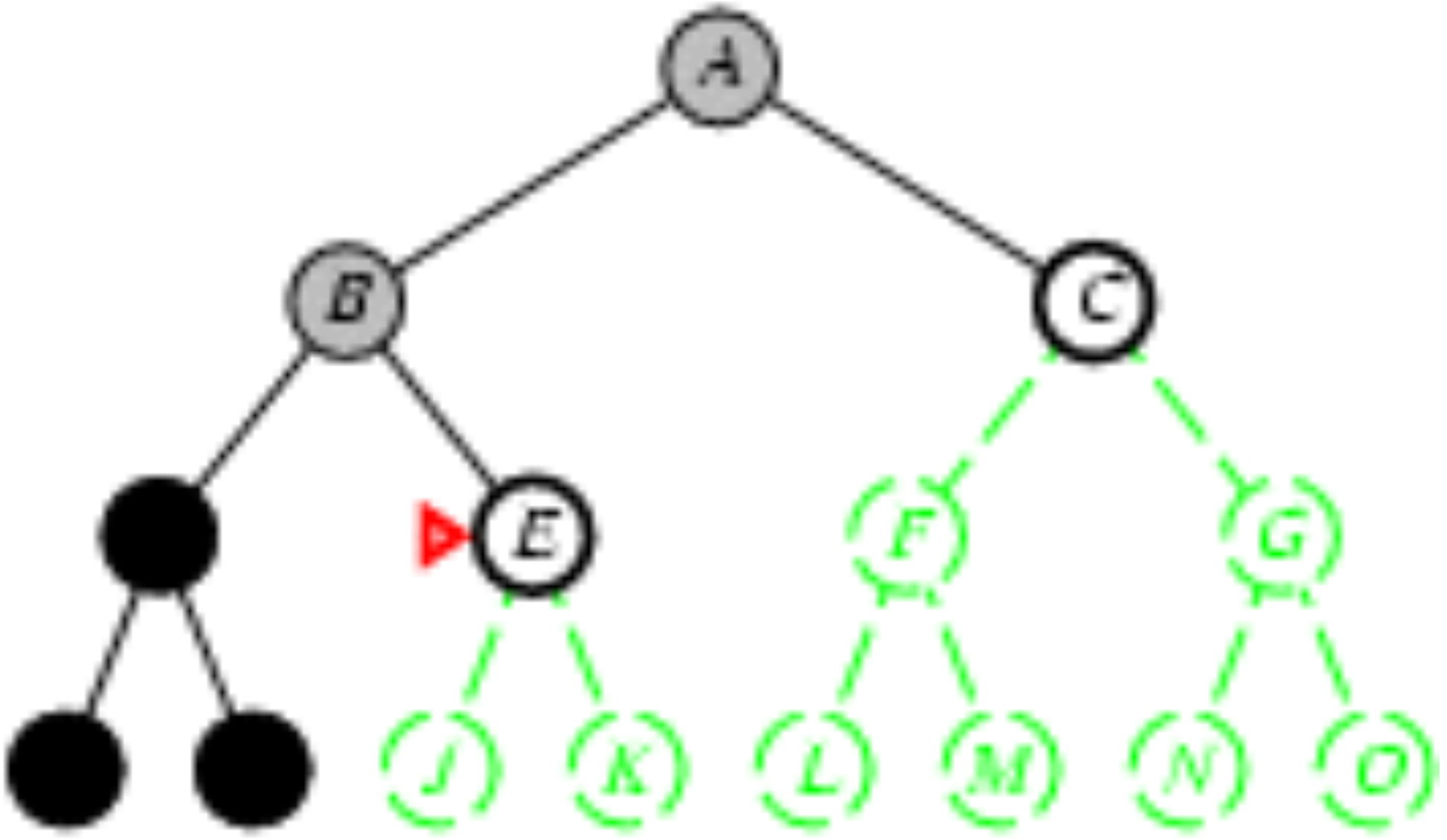


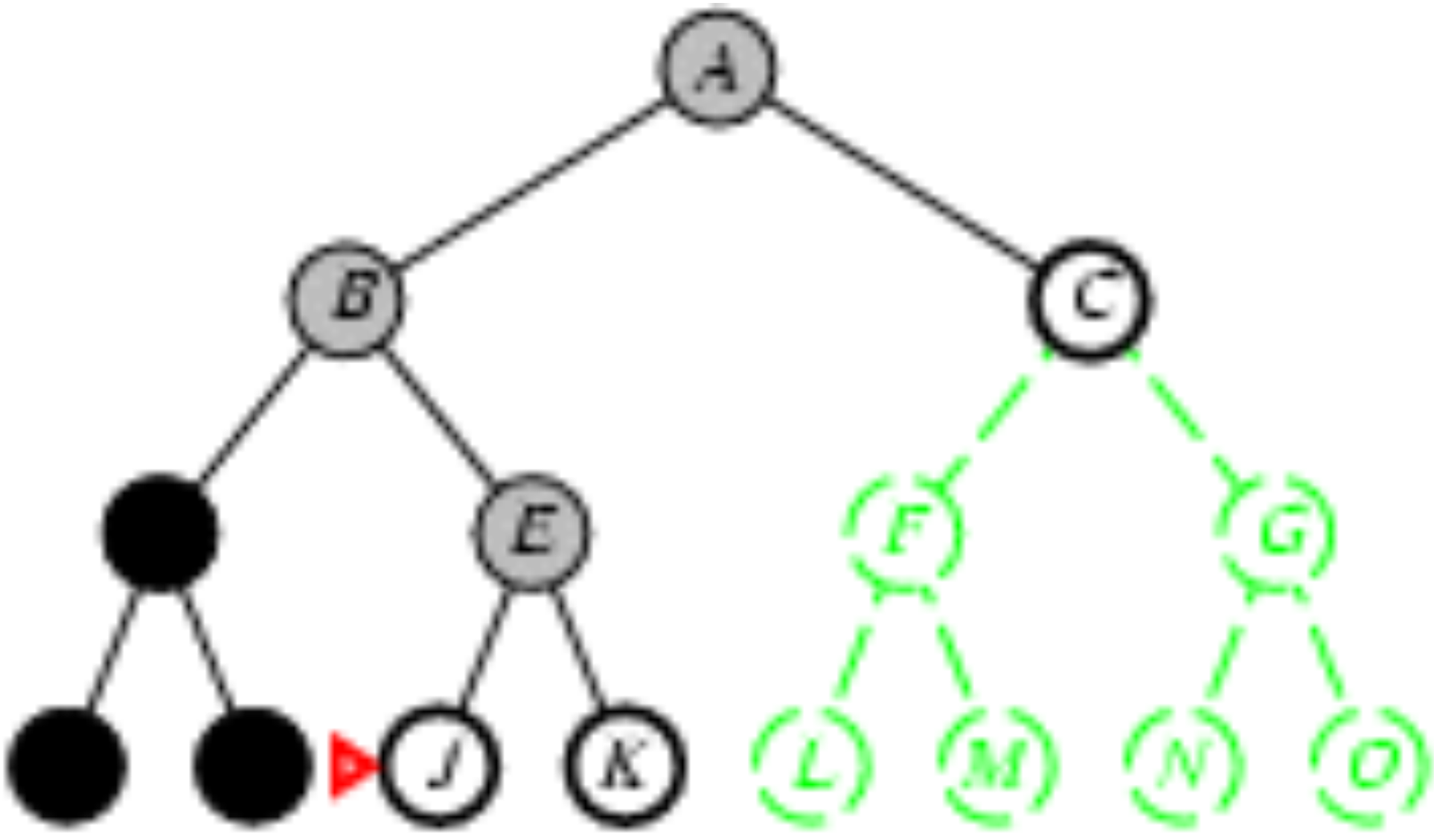

A
B
C
D
E
F
G
H
I
J
K
L
M
N
O

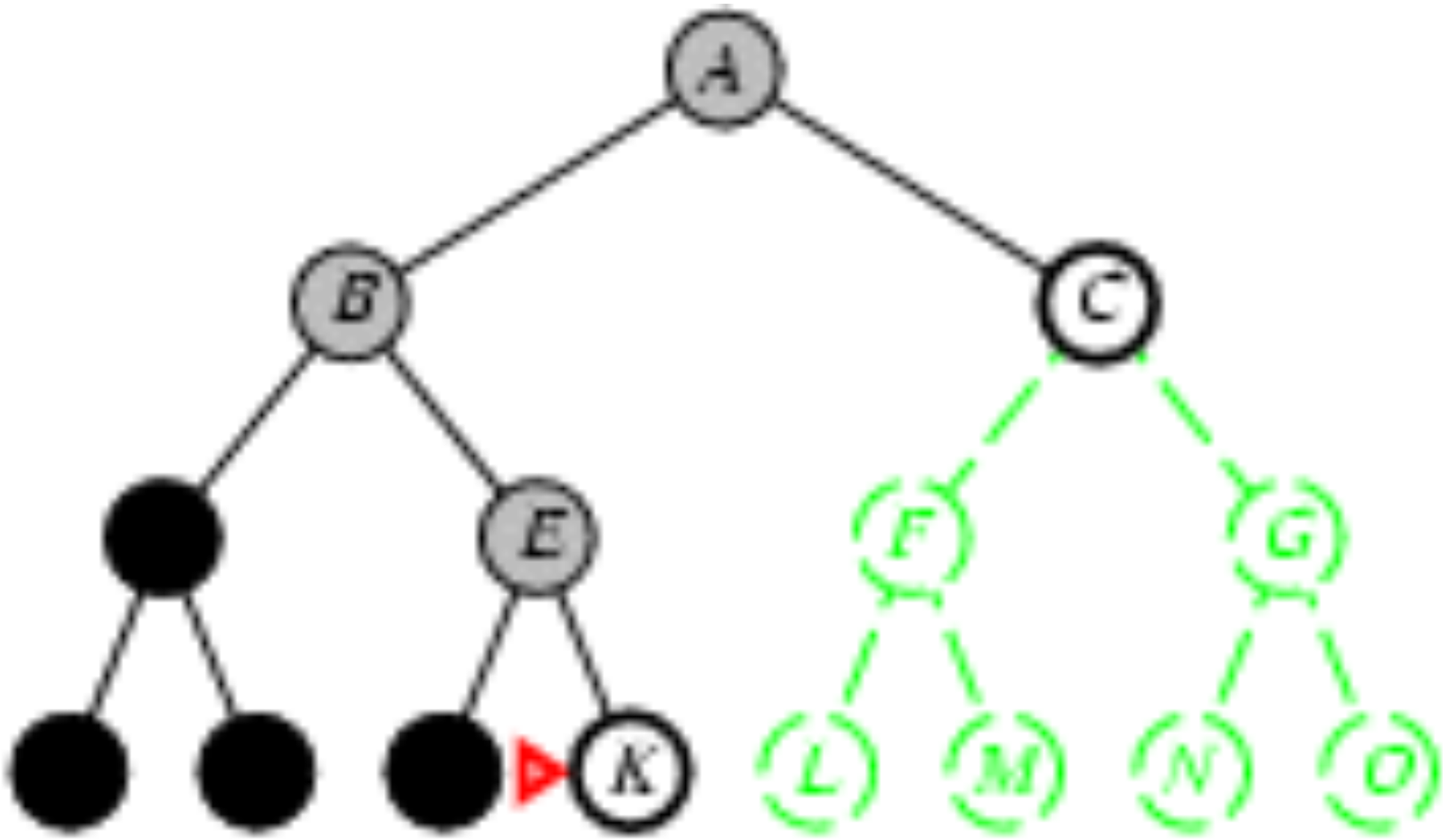

A
B
C
D
E
F
G
H
I
J
K
L
M
N
O

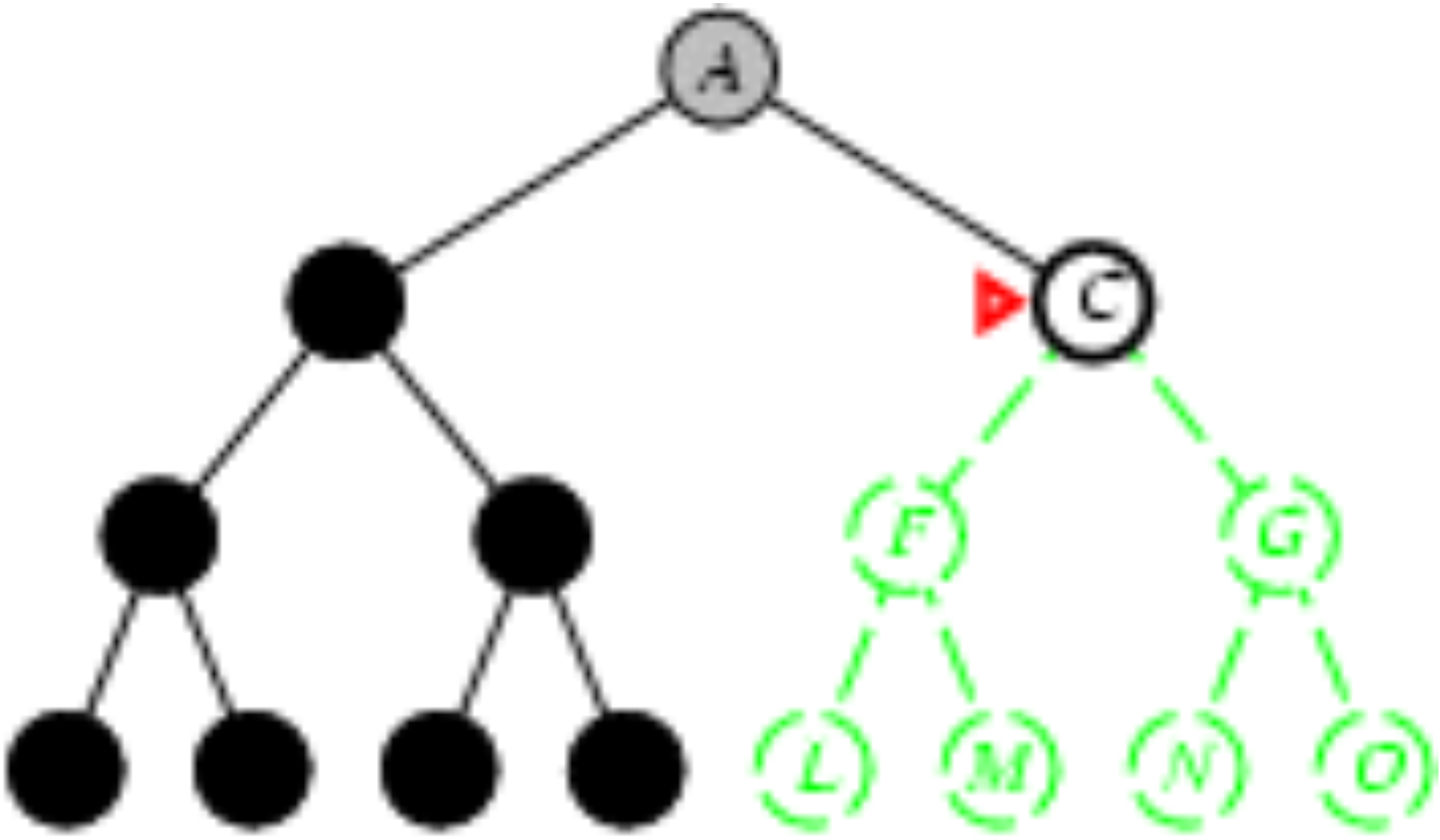

A
B
C
D
E
F
G
H
I
J
K
L
M
N
O

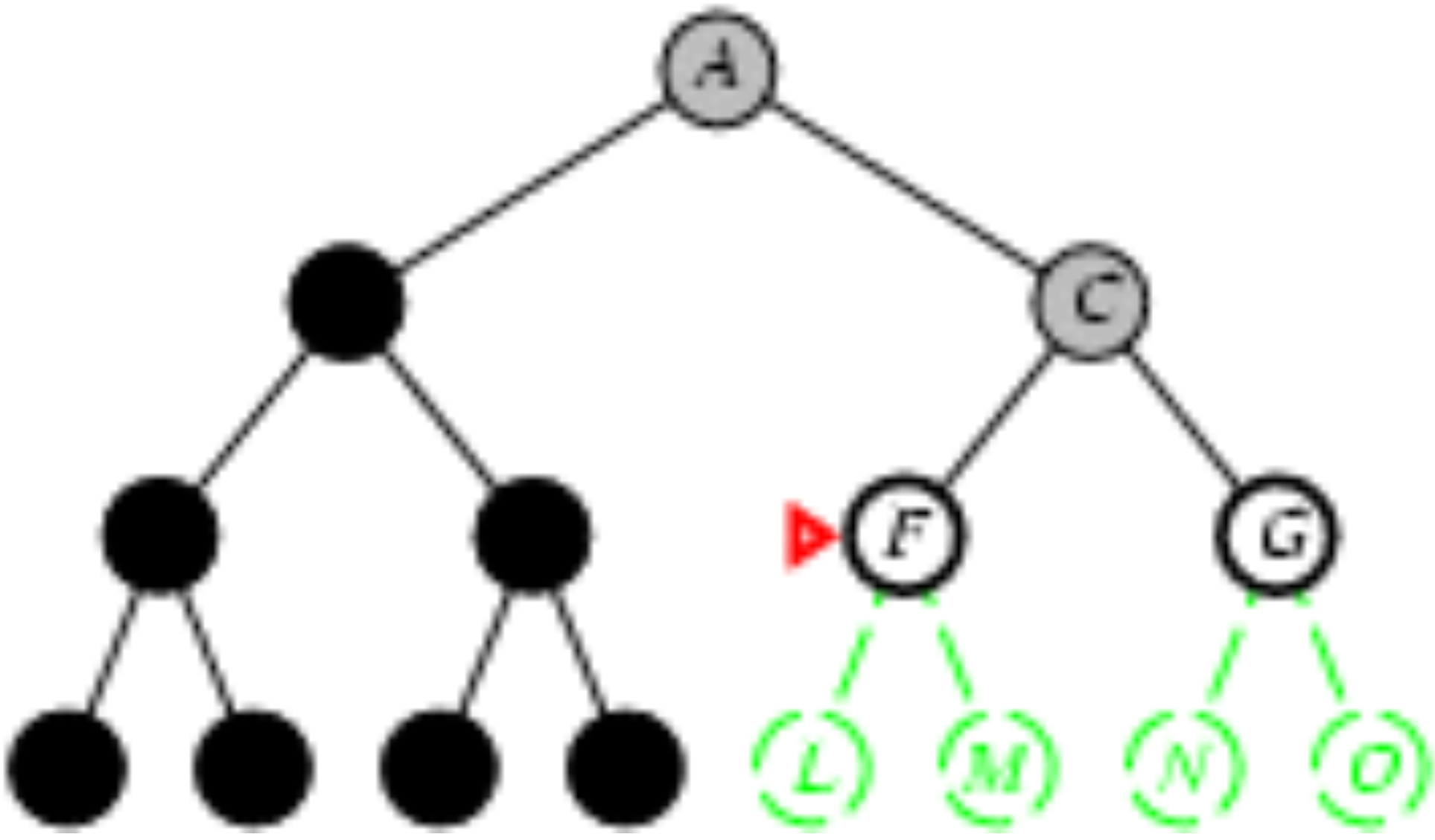

A
B
C
D
E
F
G
I
J
K
L
M
N
O


A
B
C
E
F
G
J
K
L
M
N
O


A
B
C
E
F
G
J
K
L
M
N
O


A
B
C
E
F
G
K
L
M
N
O

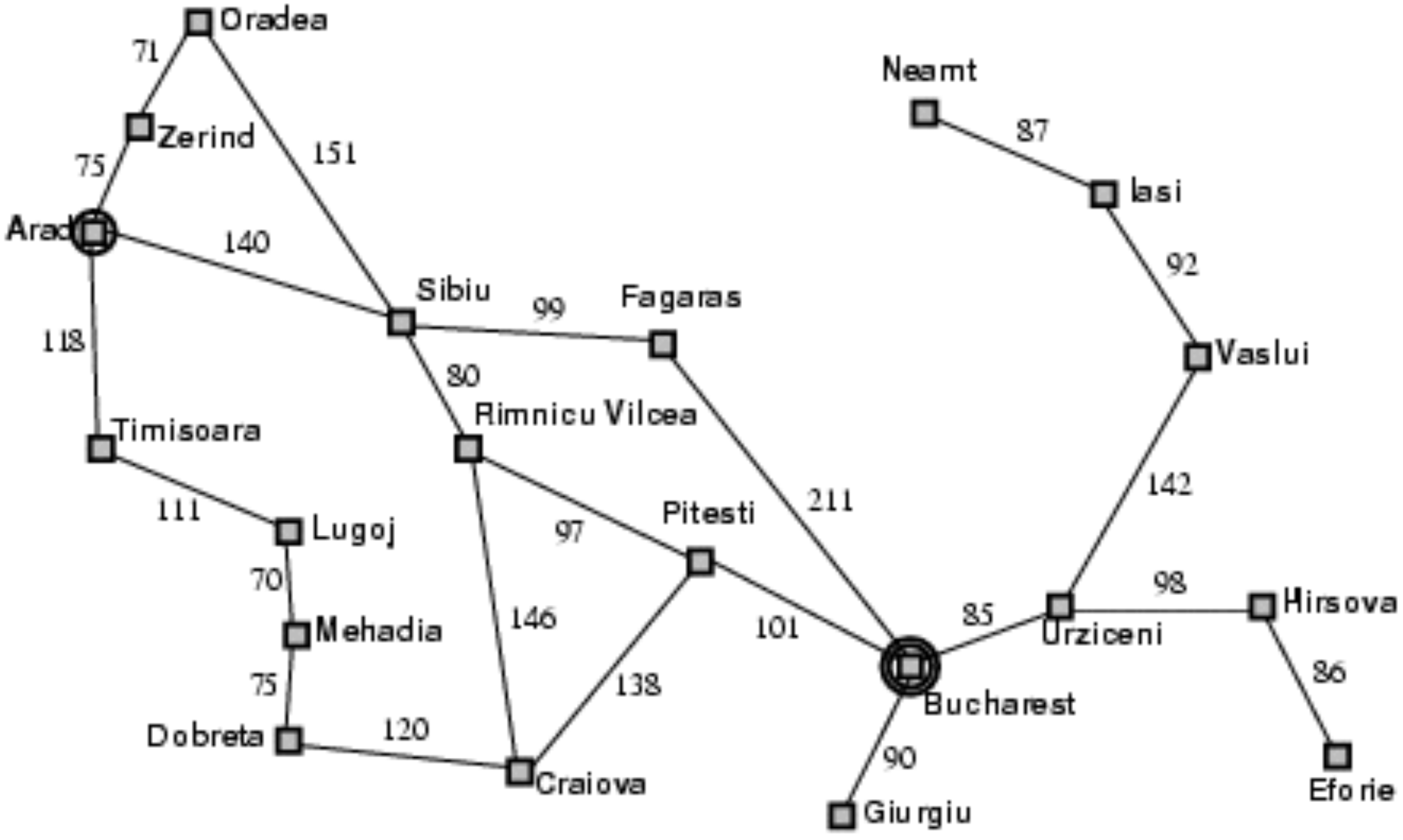

A
C
F
G
L
M
N
O


A
C
F
G
L
M
N
O

Depth-first search

- Complete? No: fails in infinite-depth spaces, spaces with loops
  - Modify to avoid repeated states along path
  - complete in finite spaces
- Time? $O(b^m)$: terrible if m is much larger than d
  - but if solutions are dense, may be much faster than breadth-first
- Space? O(bm): linear space!
- Optimal? No

From Arad to Bucharest



Depth-limited search

- Depth-first search with depth limit l; nodes at depth l have no successors

```
function DEPTH-LIMITED-SEARCH(problem, limit) returns soln/fail/cutoff
    RECURSIVE-DLS(MAKE-NODE(INITIAL-STATE[problem]), problem, limit)

function RECURSIVE-DLS(node, problem, limit) returns soln/fail/cutoff
    cutoff-occurred? ← false
    if GOAL-TEST[problem](STATE[node]) then return SOLUTION(node)
    else if DEPTH[node] = limit then return cutoff
    else for each successor in EXPAND(node, problem) do
        result ← RECURSIVE-DLS(successor, problem, limit)
        if result = cutoff then cutoff-occurred? ← true
        else if result ≠ failure then return result
    if cutoff-occurred? then return cutoff else return failure
```

Iterative deepening

- Do depth-limited search at increasing depths

function ITERATIVE-DEEPENING-SEARCH(*problem*) **returns** a solution, or failure

inputs: *problem*, a problem

for *depth* ← 0 **to** ∞ **do**
  result ← DEPTH-LIMITED-SEARCH(*problem*, *depth*)
  if *result* ≠ cutoff **then return** *result*

Summary of algorithms

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening
Complete?	Yes	Yes	No	No	Yes
Time	$O(b^{d+1})$	$O(b^{\lceil C^*/\epsilon \rceil})$	$O(b^m)$	$O(b^l)$	$O(b^d)$
Space	$O(b^{d+1})$	$O(b^{\lceil C^*/\epsilon \rceil})$	$O(bm)$	$O(bl)$	$O(bd)$
Optimal?	Yes	Yes	No	No	Yes

Benchmark problems

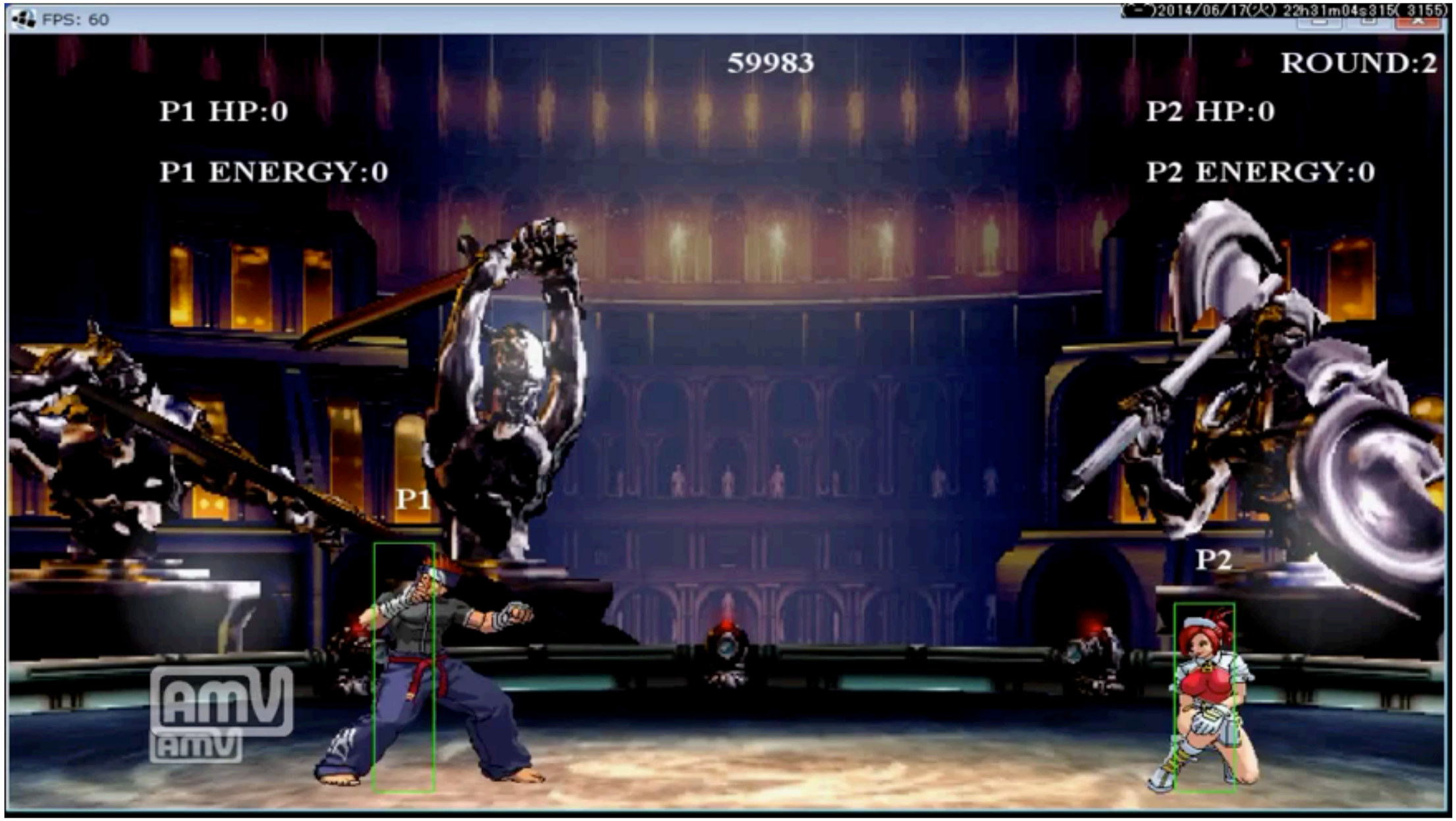
- …are problems which allow you to compare the performance of several AI algorithms
    - Should be efficient, fair and relevant
- You will choose one benchmark problem for your project

Good benchmark problems

- API
- Clear goal state and performance measure (e.g. score, number of tasks successfully completed etc)
- Solution is an action sequence
- Reasonable branching factor (or simple to discretize the action space)
- Forward model, i.e. simulation with successor function
- Fast!

Pac-Man vs Ghosts

Fighting game competition



Mario AI Competition

General Video Game Playing

- GVG-AI: a framework for *general* video game playing

- Provides a game engine for simple 2D arcade-like games

- Games are specified in VGDL (Video Game Description Language)

- Framework implemented in Java

The GVG-AI framework

gvgai.net

The GVG-AI Competition | About | Research | FAQ | Rankings | Log in | Sign up

The General Video Game AI Competition - 2015

Welcome to the General Video Game AI Competition webpage. The **GVG-AI Competition** explores the problem of creating controllers for general video game playing. How would you create a single agent that is able to play any game it is given? Could you program an agent that is able to play a wide variety of games, without knowing which games are to be played?

In this website, you will be able to participate in the General Video Game AI Competition, that will be held at GECCO and CIG 2015. This competition is run by Diego Perez, Spyridon Samothrakis, Julian Togelius, Tom Schaul and Simon Lucas. You will be able to download the starter kit for the competition and submit your controller to be in the rankings.

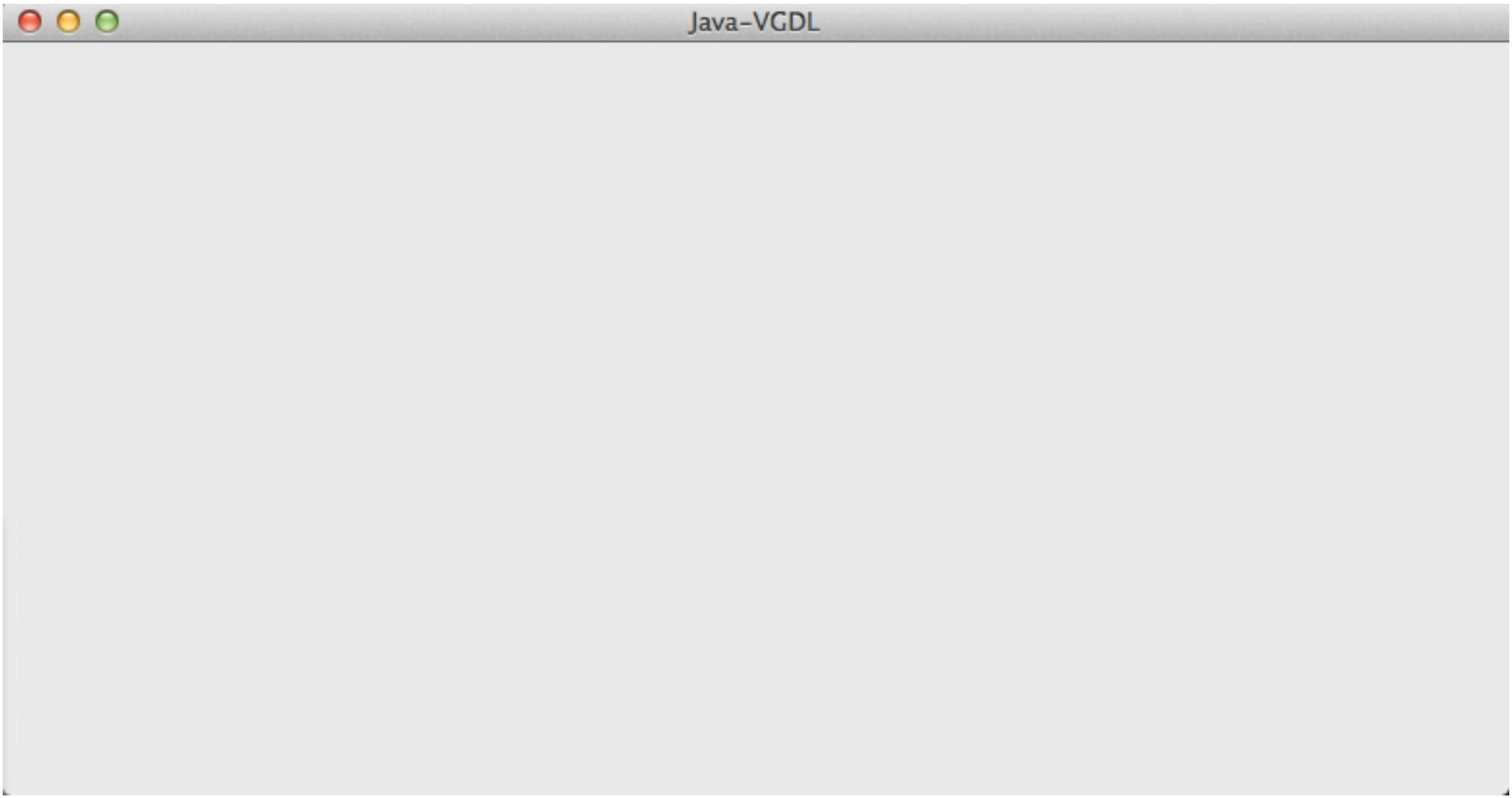
Join our **Google Group** here for the latest updates. If you are interested in studying for a **PhD** in this area, you can have a look at our Centre for Doctoral Training.

Getting Started

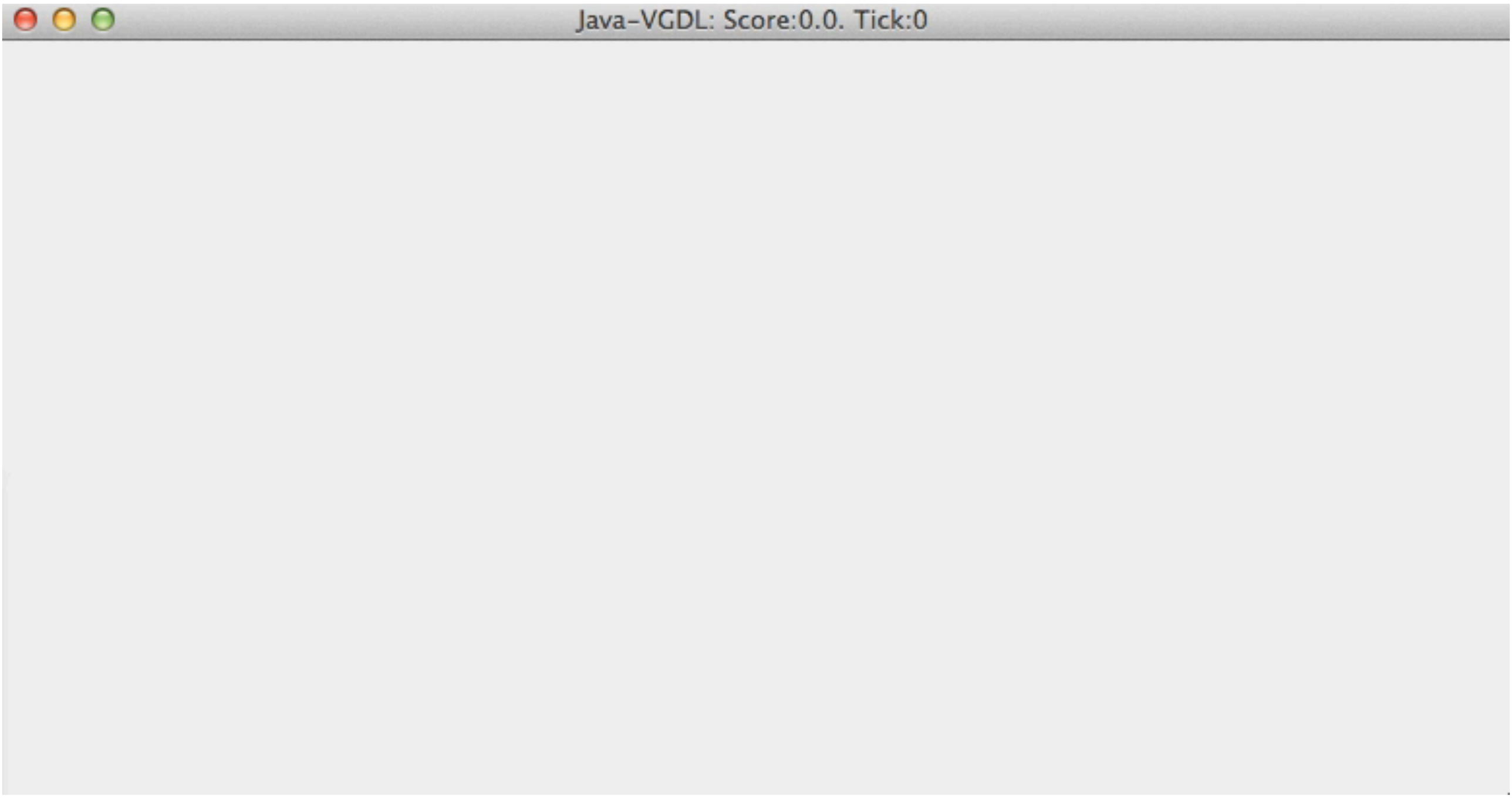
```
BasicGame
        SpriteSet
                sword > Flicker color=LIGHTGRAY limit=1 singleton=True img=sword.png
                dirt > Immovable color=BROWN img=dirt.png
                exitdoor > Door color=GREEN img=door.png
                diamond > Resource color=YELLOW limit=10 shrinkfactor=0.75 img=diamond.png
                boulder > Missile orientation=DOWN color=GRAY speed=0.2 img=boulder.png
                moving >
                        avatar  > ShootAvatar   stype=sword img=avatar.png
                        enemy > RandomNPC
                                crab > color=RED img=camel.png
                                butterfly > color=PINK img=butterfly.png
        LevelMapping
                . > dirt
                E > exitdoor
                o > boulder
                x > diamond
                c > crab
                b > butterfly
        InteractionSet
                dirt avatar > killSprite
                dirt sword  > killSprite
                diamond avatar > collectResource
                diamond avatar > killSprite scoreChange=2
                moving wall > stepBack
                moving boulder > stepBack
                avatar boulder > killIfFromAbove scoreChange=-1
                avatar butterfly > killSprite scoreChange=-1
                avatar crab > killSprite scoreChange=-1
                boulder dirt > stepBack
                boulder wall > stepBack
                boulder diamond > stepBack
                boulder boulder > stepBack
                enemy dirt > stepBack
                enemy diamond > stepBack
                crab butterfly > killSprite
                butterfly crab > transformTo stype=diamond scoreChange=1
                exitdoor avatar > killIfOtherHasMore resource=diamond limit=9

TerminationSet
        SpriteCounter stype=avatar limit=0 win=False
        SpriteCounter stype=exitdoor limit=0 win=True
```

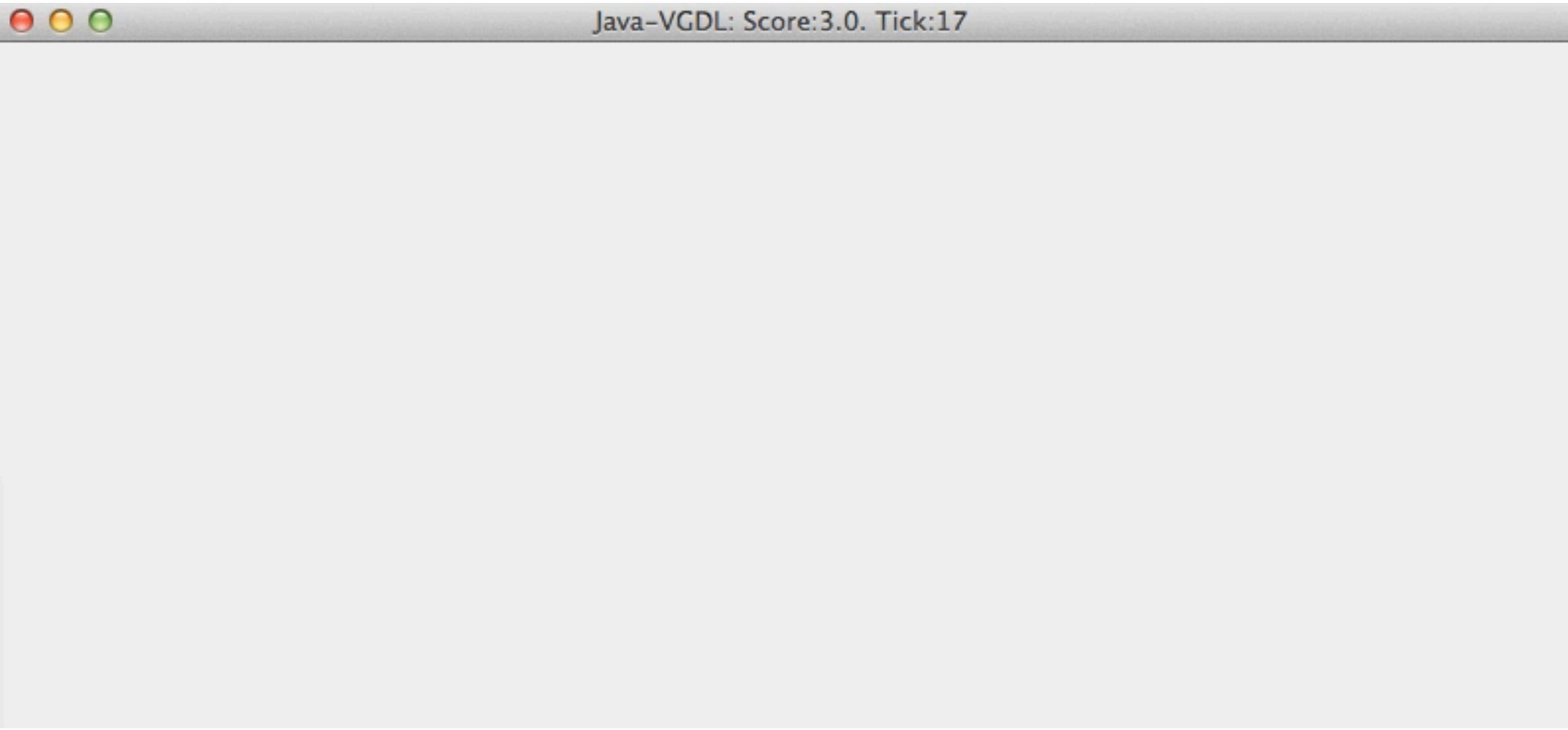

Java-VGDL



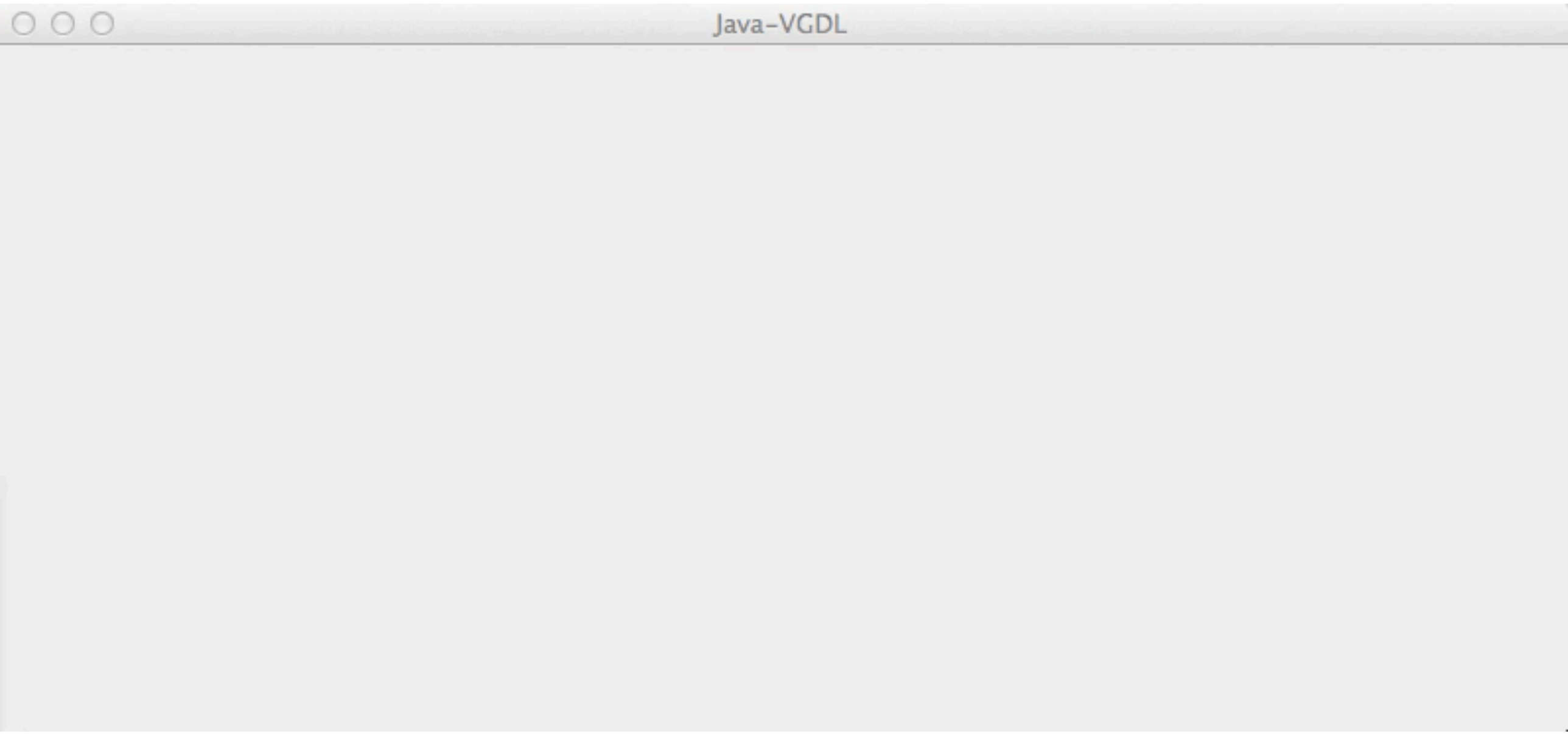
Human player in Boulder Dash



Random controller on Boulder Dash



Random controller on “Aliens” (Space Invaders)



MCTS controller on “Aliens” (Space Invaders)

The project

- Part 1: Implement *all* of the algorithms in the course (those that are applicable; I will provide lists) on your benchmark; do a thorough comparison of them
  - Highly valued: clever implementations
  - Highly valued: good evaluations, well reported
  - Highly valued: good explanations of the differences in performance between algorithms
  - This is also your homework

The project

- Part 2: Do something creative. For example:
  - Develop a hybrid algorithm that combines elements of several algorithms
  - Come up with a new algorithm (maybe based on one of the existing ones)
  - Use the algorithms for another purpose, for example tune the game, generate new robots…

The report

- Maximum 10 pages
- Should be well written
- Part 1 should be well reported
- Part 2 should also be well motivated

Submission deadline:

Friday, December 4

Sorry!