中国地质大学
China University of Geosciences

# 第三章

# 通过搜索进行问题求解

计算机学院　人工智能B课程

主讲人　赵曼

Lecturer：Zhaoman, Introduce to AI,
Department of Computer Science, CS, CUG

中国地质大学 China University of Geosciences

# 目 录



中国地质大学 China University of Geosciences

# 目 录



# Problem solving in AI 人工智能中的问题求解

- **The solution** 解

  is a sequence of actions to reach the goal.

  是一个达到目标的动作序列。

- **The process** 过程

  look for the sequence of actions,which is called search.

  寻找该动作序列，称其为搜索。

- **Problem formulation** 问题形式化

  given a goal, decide what actions and states to consider.

  给定一个目标，决定要考虑的动作与状态。

- **Why search** 为何搜索

  Some NP-complete or NP-hard problems, can be solved only by search.

  对于某些NP完或者NP难问题，只能通过搜索来解决。

- **Problem-solving agent** 问题求解智能体

  is a kind of goal-based agent to solve problems through search.

  是一种基于目标的智能体，通过搜索来解决问题。

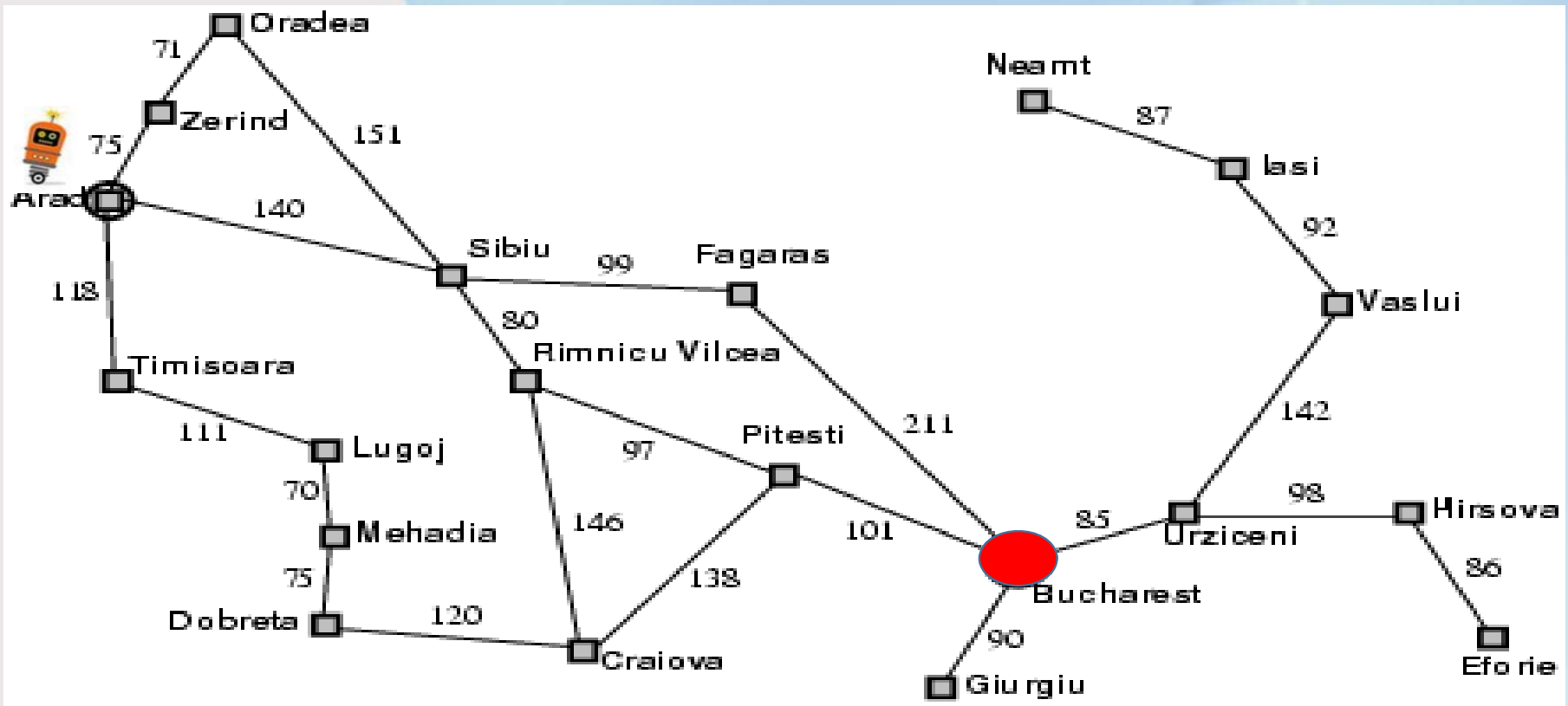
# Algorithm of Simple Problem Solving Agents 简单的问题求解智能体算法

**function** SIMPLE-PROBLEM-SOLVING-AGENT(*percept*) **return** an action

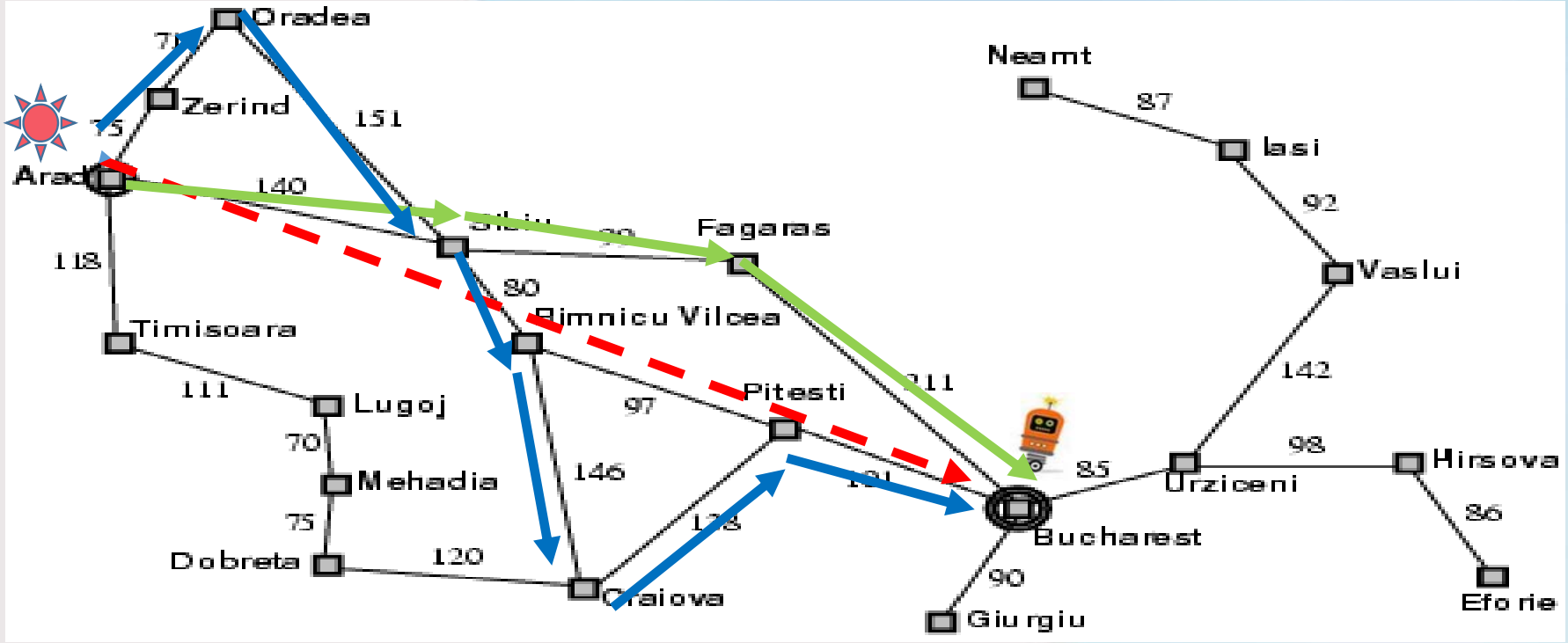
**static**: *seq*, an action sequence, initially empty
*state*, some description of the current world state
*goal*, a goal, initially null
*problem*, a problem formulation
action, the most recent action, initially none

*state* ← UPDATE-STATE(*state*, *percept*)
**if** *seq* is empty **then**
*goal* ← FORMULATE-GOAL(*state*)
*problem* ← FORMULATE-PROBLEM(*state*, *goal*)
*seq* ← **SEARCH**(*problem*)
**if** *seq*= failure **then** **return**a null action
*action* ← FIRST(*seq*)
*seq* ← REST(*seq*)
return *action*

# 例子:罗马尼亚Romania度假问题



# 例子:罗马尼亚Romania度假问题



# Related Terms 相关术语

## State space 状态空间

The state space of the problem is formally defined by: Initial state, actions and transition model.

问题的状态空间可以形式化地定义为：初始状态、动作和转换模型。。

## Graph 图

State space forms a graph, in which nodes are states, and links are actions.

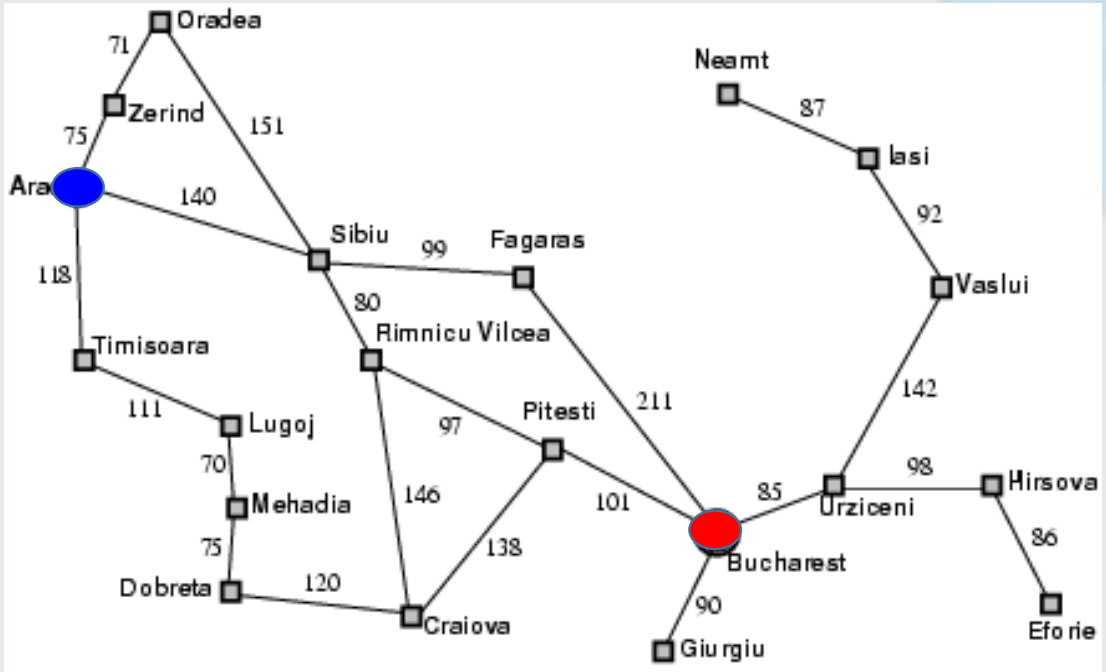
状态空间 形成一个图，其中节点表示状态、链接表示动作 。

## Path 路径

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

状态空间的一条路径是由一系列动作连接的一个状态序列

# Five Items to Formulate a Problem 形式化的五个要素




## 1) Initial state 初始状态

The agent starts in.

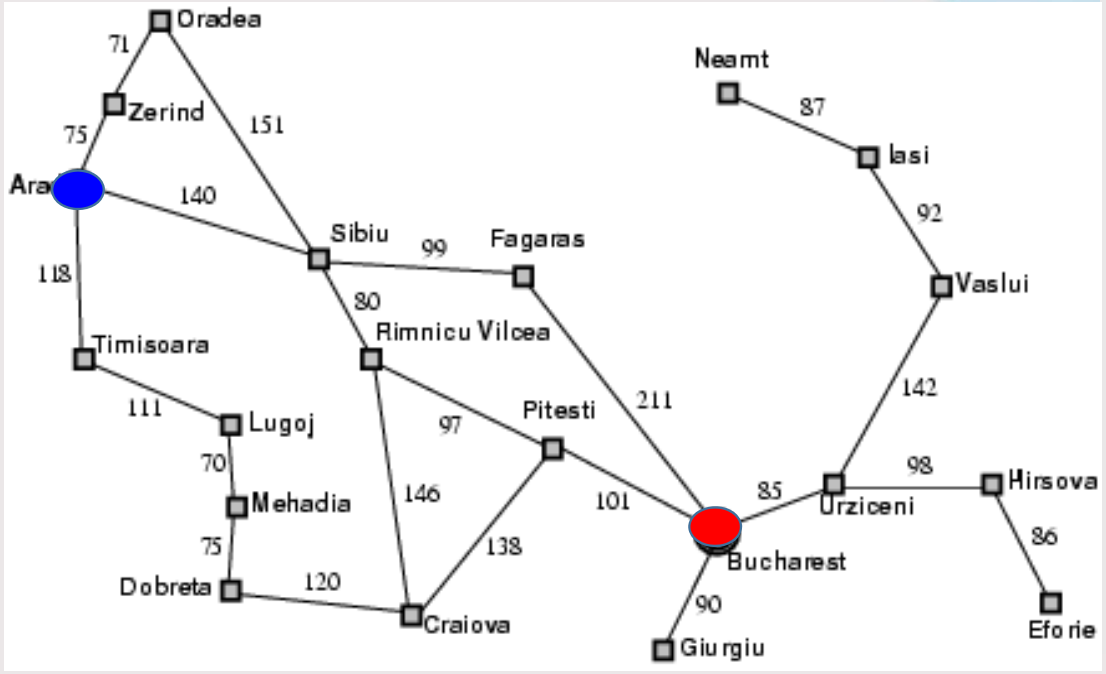
即智能体出发时的状态 。

E.g., the initial state for the agent in Arad may be described as:

例如，该智能体位于Arad的初始状态可以记作：

*In(Arad).*

# Five Items to Formulate a Problem 形式化的五个要素




## 2) Actions 动作

- A description of the possible actions available to the agent.

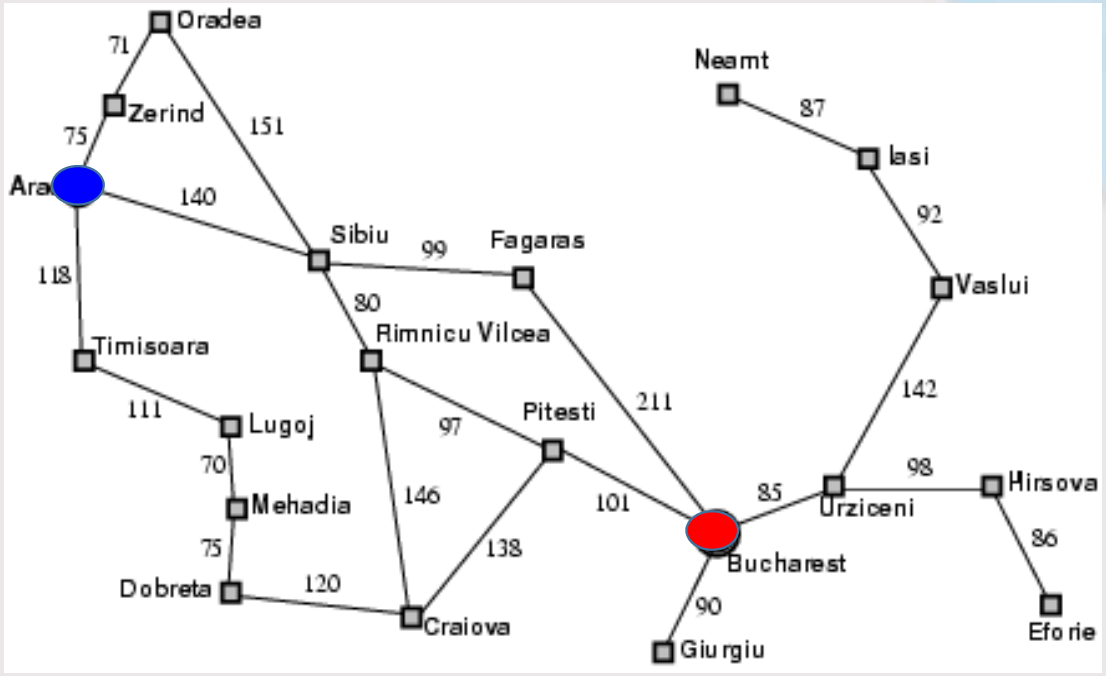
  描述该智能体可执行的动作。

- **ACTION(s)** returns the actions that can be executed in s. E.g.,

  **ACTION(s)**返回s状态下可执行的动作序列。例如，

  *{Go(Zerind), Go(Sibiu), Go(Timisoara)}.*

# Five Items to Formulate a Problem 形式化的五个要素




## 3) Transition model 转换模型（后继函数）

- A description of what each action does.

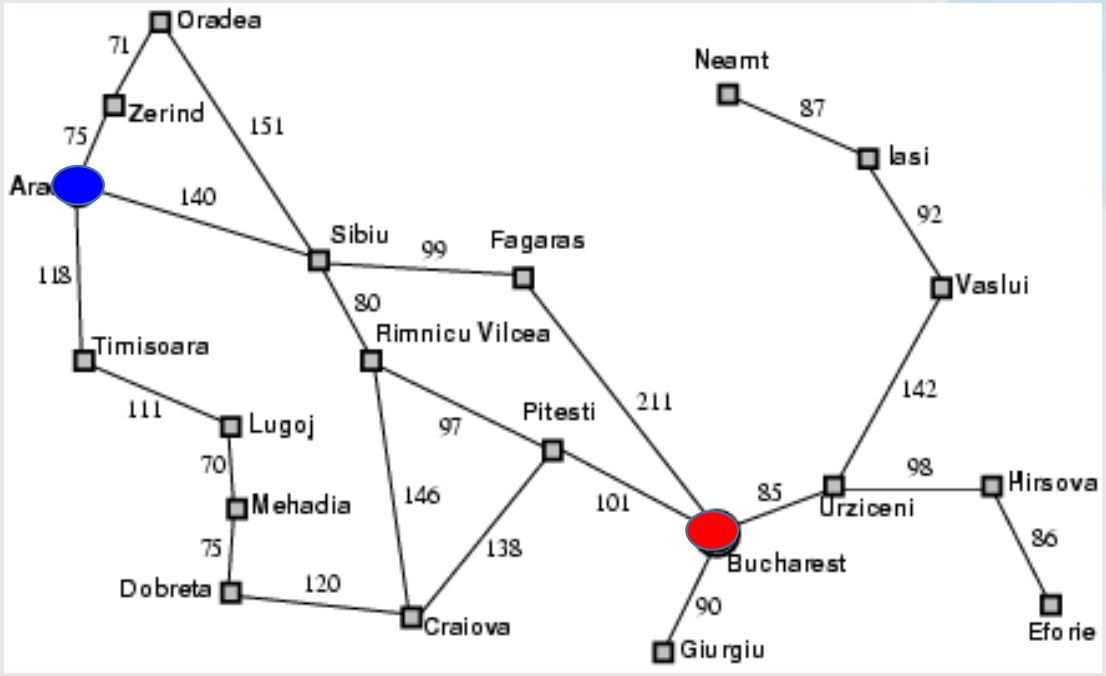
  描述每个动作做什么。。

- RESULT(s, a) returns the state from doing action a in s .. E.g.,

  RESULT(s, a) 返回s， 动作a之后的状态。例如，

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

# Five Items to Formulate a Problem 形式化的五个要素




## 4) Goal test 目标测试

- To determine whether a given state is a goal state.

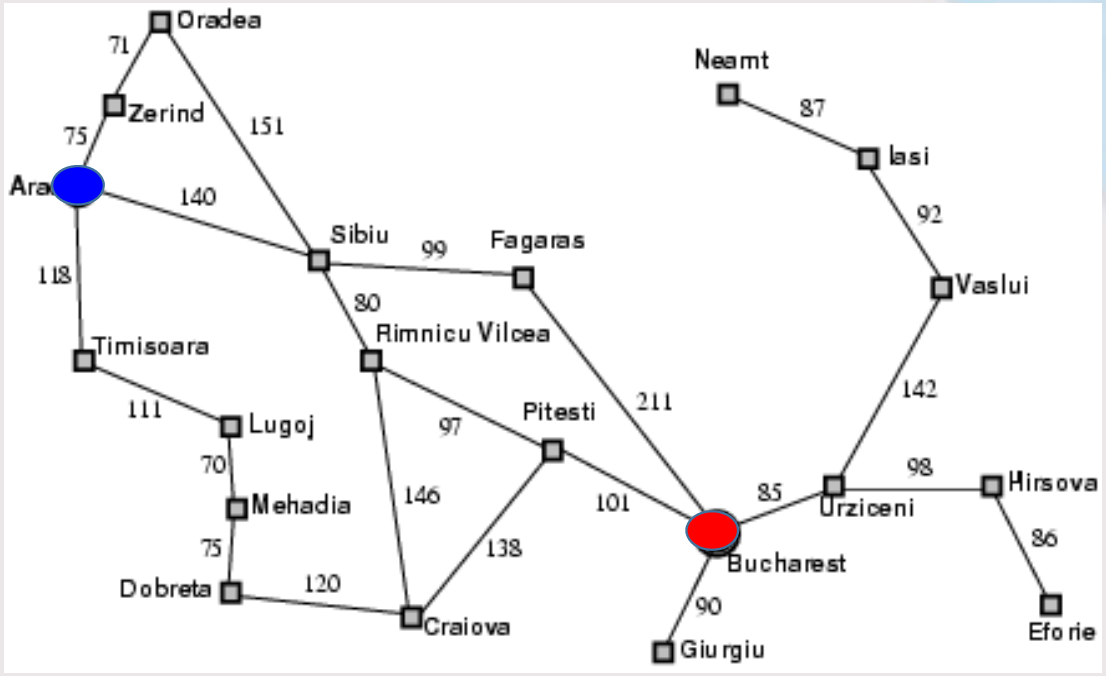
确定一个给定的状态是否是目标状态。

E.g., the agent's goal in Bucharest is the singleton set:

例如：智能体在 Bucharest的目标是单元素集合：

{*In*(*Bucharest*)}.

# Five Items to Formulate a Problem 形式化的五个要素




## 5) Path cost 路径代价

- To assign a numeric cost to each path.

即每条路径所分配的一个数值代价。

E.g., step cost of taking action a in state s to reach state s' is denoted by:

例如：状态s下执行动作a到达状态s'的步骤代价表示：

$c(s, a, s')$.

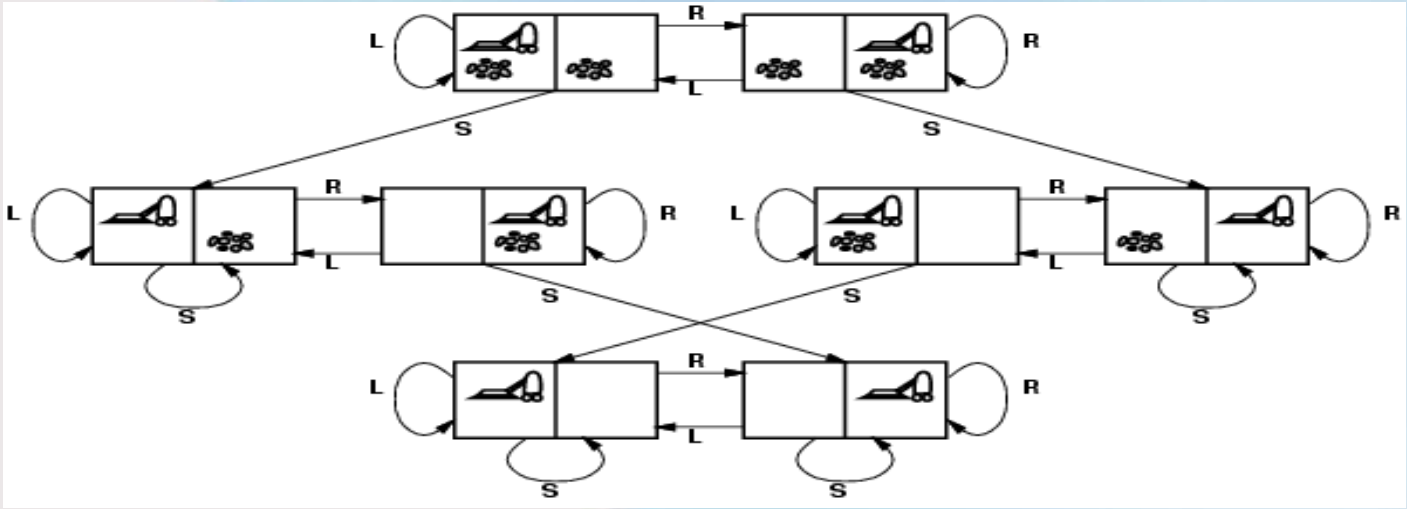
中国地质大学 China University of Geosciences

# 目 录

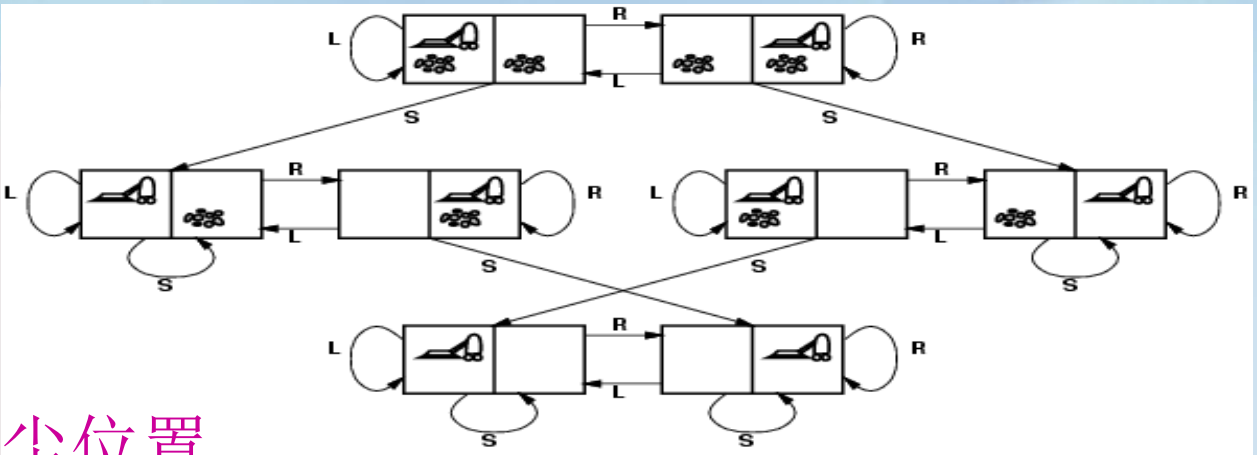


# 真空吸尘器世界状态空间图

- 状态?
- 动作?
- 转换模型?
- 目标测试?
- 路径代价?

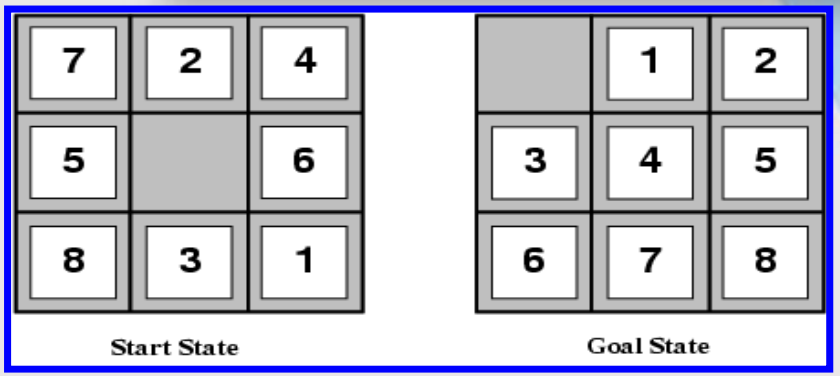


# 真空吸尘器世界状态空间图




- 状态：机器人的位置及灰尘位置
- 动作： 向左，向右，吸
- 转换模型：该动作应有的预期效果的预期效果（除无效动作）
- 目标测试：所有位置都干净
- 路径代价：每做一个动作计代价为1

# 滑块游戏: 八数码问题



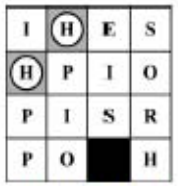

The 8 -puzzle belongs to the family of sliding block puzzles, this family is known be NP - complete.
8数码难题属于滑块数码难题家族，
这个难题家族，这个被 认为是 NP-完的。

8-puzzle: 3×3 board with 8 numbered tiles and a blank space.
8数码难题：3×3棋盘上有8个数字棋子和一个空格。
A tile adjacent to the blank space can slide into the space. The object is to reach a specified goal state.与空格相邻的滑块可以移向该空格，目的是达到一个指定的目标状态。


3x3 sliding puzzle.


7x7 sliding block puzzle

15-puzzle

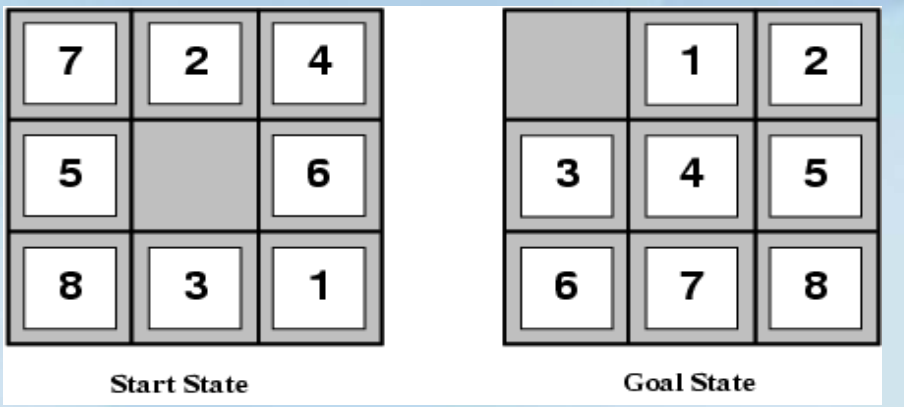


Word puzzle

华容道

# 滑块游戏: 八数码问题

- 状态?
- 动作?
- 转换模型？
- 目标测试?
- 路径代价?




- 状态： 8个数字块的放置情况
- 动作： 空格向上，向下，向左，向右移动
- 转换模型： 移动后，产生数字块和空格的互换
- 目标测试： 目标状态（给定的）
- 路径代价： 每移动一步代价为1

# $(N^2-1)$-数码的问题空间与运行时间：

- 8-puzzle → 9! = 362,880 states
- 15-puzzle → 16! ~ 2.09 x $10^{13}$ states
- 24-puzzle → 25! ~ $10^{25}$ states

实际可达到的空间为**1/2**


8-puzzle → 362,880 states

0.036 sec

15-puzzle → 2.09 x $10^{13}$ states

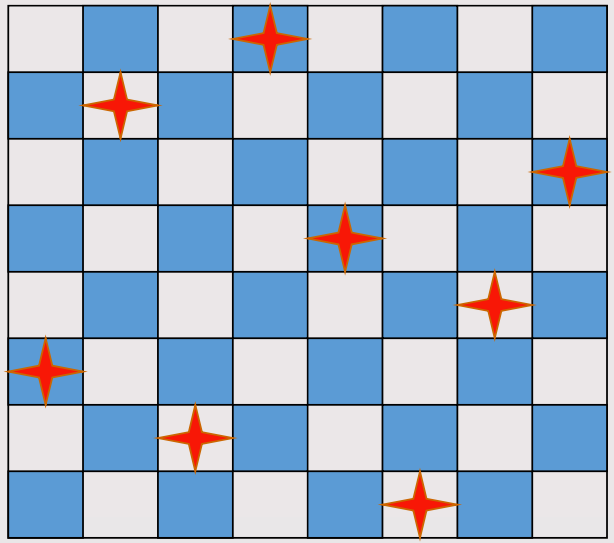
~ 55 hours

24-puzzle → $10^{25}$ states

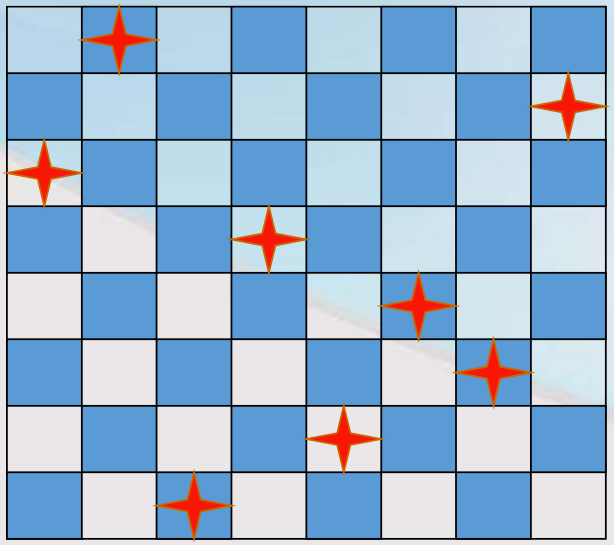
> $10^9$ years

100 millions states/sec

# 八皇后问题



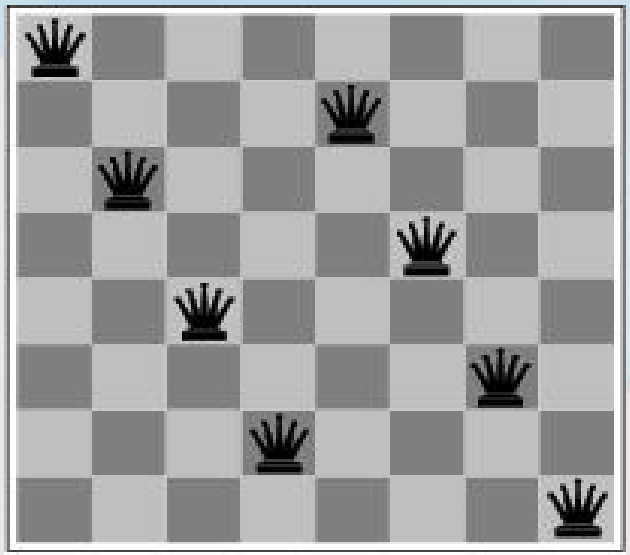
一个解



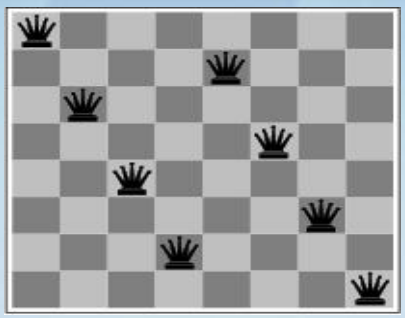
不是一个解

# 八皇后问题

- 状态??
- 初始状态??
- 行动??
- 转换模型??
- 目标测试??
- 路径代价??

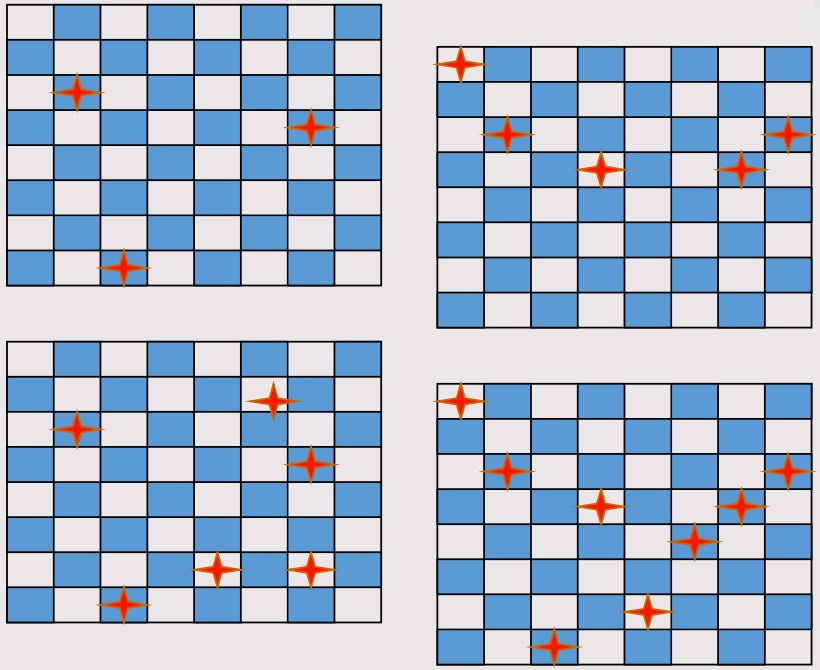


# 八皇后问题



| | 增量形式 | vs. | 完全状态形式化 |
|---|---|---|---|
| • 状态?? | 棋盘上皇后的摆放位置 | | |
| • 初始状态?? | 空 | vs. | 8个放满 |
| • 行动?? | 放置一个棋子 | vs. | 移动一个棋子 |
| • 转换模型?? | 将一个棋子在空格 | vs. | 移动一个棋子到另一空格 |
| • 目标测试?? | 所有皇后互相不攻击 | | |
| • 路径代价?? | 无意义（或每步代价为1） | | |

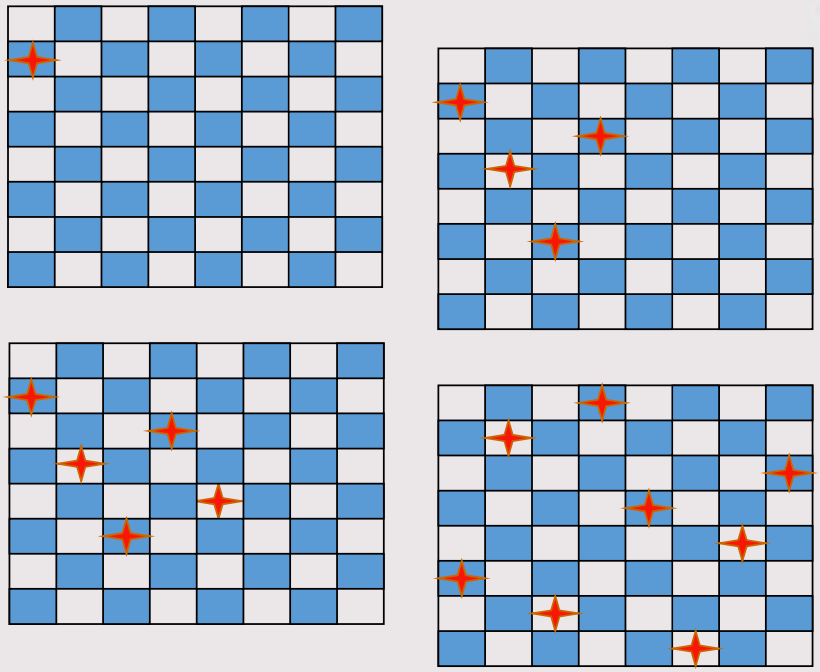
# 八皇后问题：增量形式化-1



- 状　　态: 0~8个皇后摆放在棋盘上
- 初始状态: 空棋盘
- 后继函数: 把一个皇后放在棋盘上的任一位置
- 路径耗散: 无关
- 目标测试: 8个皇后无冲突的放在棋盘上

→ $64 \times 63 \times \ldots \times 57 \sim 3 \times 10^{14}$ states

# 八皇后问题：增量形式化-2



→ 2,057 states

- 状态：摆放k个皇后（0≤k≤8）的安排，要求最左边k列里每列一个皇后，保证没有一个皇后能相互攻击；
- 初始状态：盘上没有皇后
- 模型转换：把一个皇后添加到最左边空列的任何格子内，只要该位子不被攻击；
- 路径耗散：无意义
- 目标：8个皇后都在盘上

## N皇后问题启示：

- **特点：**一个解是最终的节点而不是一条路径；
- **状态空间的数目：**100个皇后问题初始状态有$10^{400}$个状态，改进后为$10^{52}$，但仍然是很大；
- 但是技术上是存在解决n很大的皇后问题；
- 通过对状态空间分布的研究可以很好的解决。

# 问题求解智能体

四个基本步骤：

1、目标形式化（Goal formulation）

- 成功的状态描述

2、问题形式化（Problem formulation）

- 根据所给的目标考虑行动和状态的描述

3、搜索（Search）

- 通过对行动序列代价计算来**选取**最佳的行动序列.

4、执行（Execute）

- 给出“解”执行行动.

**问题求解：形式化——搜索——执行**

# 问题形式化

- 一个问题可以由四个组成部分来形式化：
  1、初始状态
  2、动作
  3、转换模型或后续函数
  4、目标测试
  5、路径代价
- 一个解是从初始状态到目标状态的动作系列

  解、代价、耗散值、最优解、无解……

# 目 录



# 3.1 问题求解

- 我们怎么找到解决问题的方法？
  - 搜索状态空间（记住空间的复杂性取决于状态表示）
    - 通过显式生成树搜索
      - ✓根=初始状态。
      - ✓节点和叶子通过后继函数生成。
    - 在一般的搜索生成图（相同的状态，通过多条路径）

- **基本思想:**

  通过产生已经探索到的状态的**后续状态**的方法来**离线**地进行状态空间的**模拟**搜索。

# 3.2 树搜索方法

基本思想：

- 通过产生已经探索到的状态的后续状态的方法来离线地进行状态空间的模拟搜索。

**function TREE-SEARCH(*problem, strategy*) return a solution or failure**

**Initialize search tree to the *initial state* of the *problem***

**loop do**

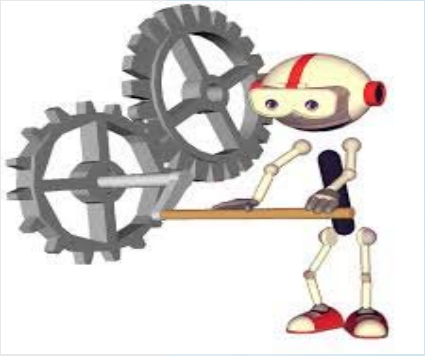
**if no candidates for expansion then return *failure***

**choose leaf node for expansion according to *strategy***

**if node contains goal state then return *solution***

**else expand the node and add resulting nodes to the search tree**

**enddo**



# 罗马尼亚问题的树搜索过程

**function TREE-SEARCH(*problem, strategy*) return a solution or failure**

**Initialize search tree to the *initial state* of the *problem***

**do**

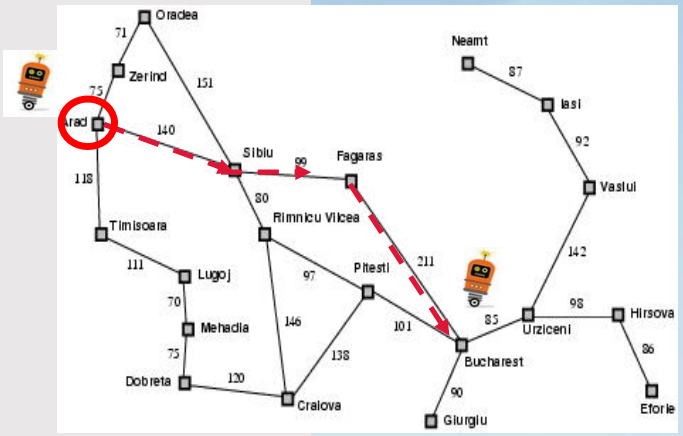
**if no candidates for expansion then return *failure***

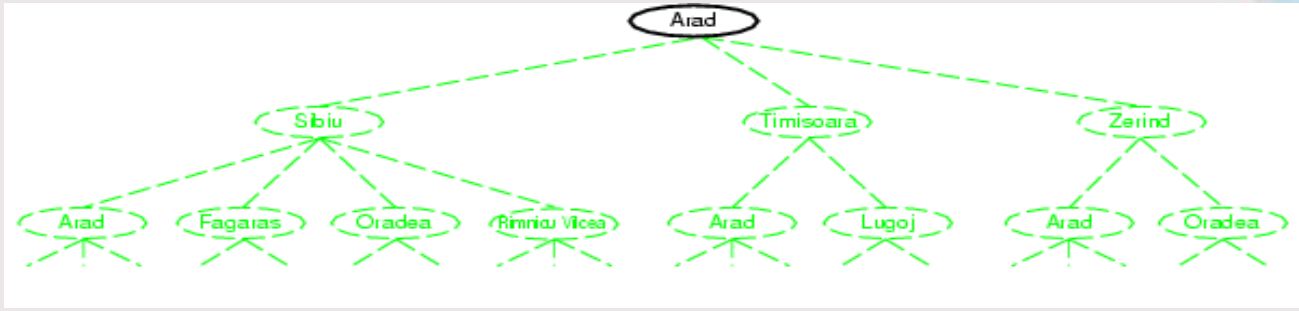
**choose leaf node for expansion according to *strategy***

**if node contains goal state then return *solution***

**else expand the node and add resulting nodes to the search tree**

**enddo**





注：

阴影：表示该节点已被扩展。

粗实线：表示该节点已被生成，但尚未扩展。

浅虚线：表示该节点尚未生成。

# 罗马尼亚问题的树搜索过程

**function** TREE-SEARCH(*problem, strategy*) **return** a *solution* or *failure*

Initialize search tree to the *initial state* of the *problem*

**do**

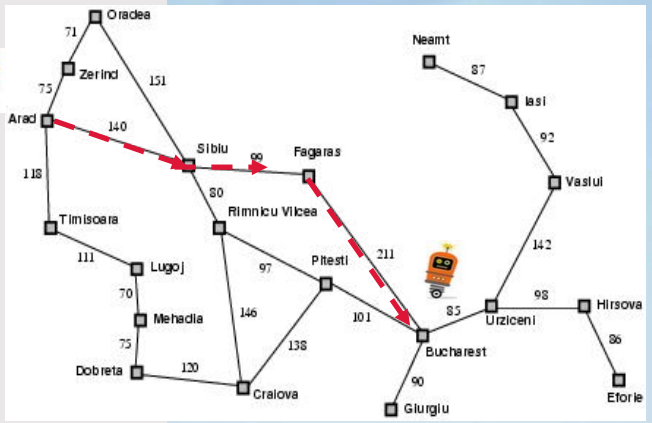
**if** no candidates for expansion **then return** *failure*

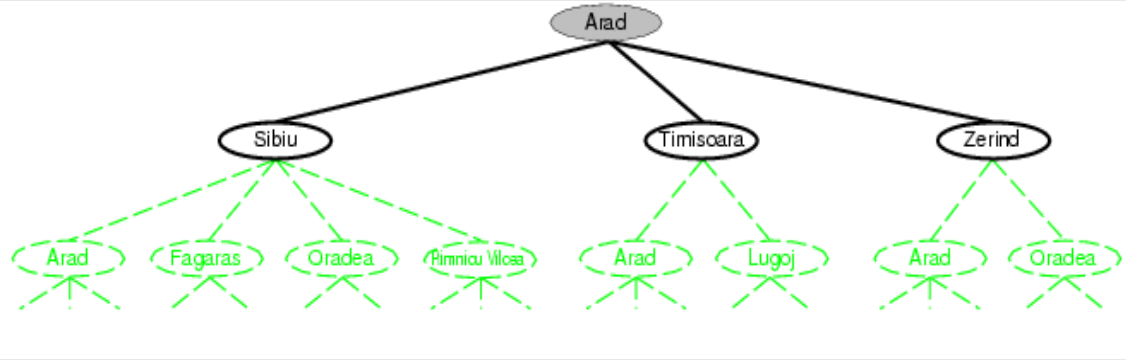
choose leaf node for expansion according to *strategy*

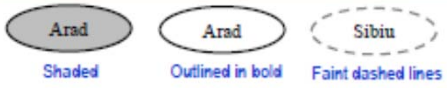
**if** node contains goal state **then return** *solution*

**else** expand the node and add resulting nodes to the search tree

**enddo**




注：
阴影：表示该节点已被扩展。
粗实线：表示该节点已被生成，但尚未扩展。
浅虚线：表示该节点尚未生成。

# 罗马尼亚问题的树搜索过程

**function** TREE-SEARCH(*problem, strategy*) **return** a solution or failure

Initialize search tree to the *initial state* of the *problem*

**do**

**if** no candidates for expansion **then return** *failure*

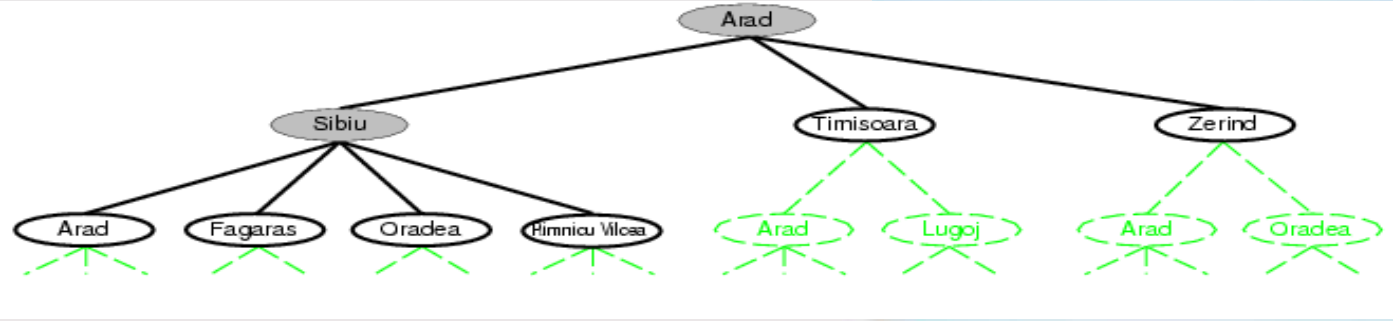
choose leaf node for expansion according to *strategy*

**if** node contains goal state **then return** *solution*

**else** expand the node and add resulting nodes to the search tree

**enddo**

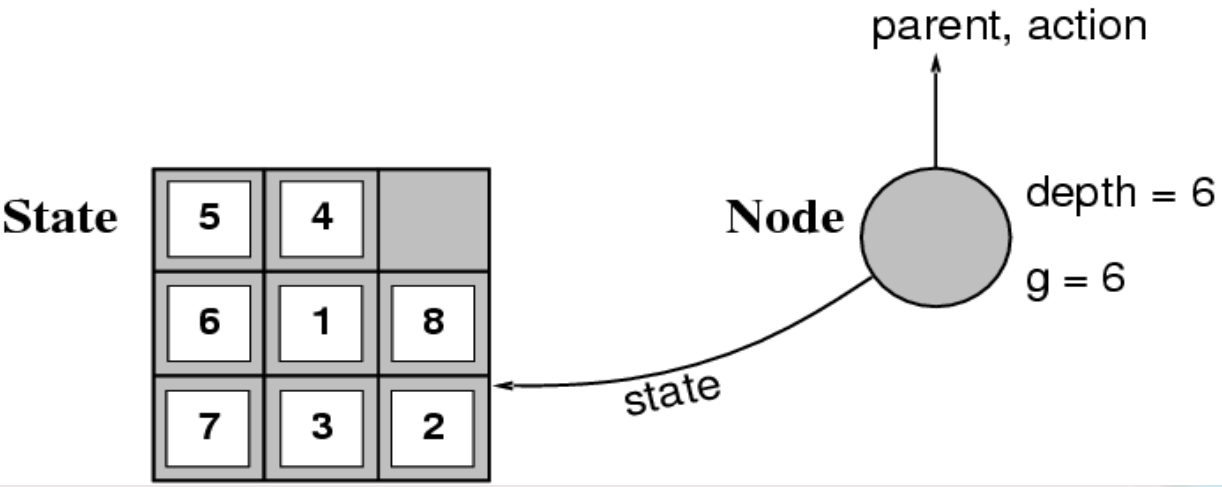
**Determines search process!!**



# 状态与节点

- 一个状态是一个物理配置的表示，而一个节点是一个数据结构，它的组成部分。

  它包含：状态、父亲节点、动作、路径代价和深度



# 树搜索算法实现:

**function TREE-SEARCH(*problem,fringe*) return a solution or failure**

*fringe* ← INSERT(MAKE-NODE(INITIAL-STATE[*problem*]), *fringe*)

**loop do**

**if** EMPTY?(*fringe*) **then return failure**

*node* ← REMOVE-FIRST(*fringe*)

**if** GOAL-TEST[*problem*] applied to STATE[*node*] succeeds

**then return SOLUTION(*node*)**

*fringe* ← INSERT-ALL(**EXPAND**(*node*, *problem*), *fringe*)

注：该 *fringe*（亦称*open list*）：一种数据结构，用于存储所有的叶节点。

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

# 3.3 通用图搜索算法

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

initialize the ***fringe*** using the initial state of *problem*

initialize the ***explored*** to be empty

**loop do**

**if** the *fringe* empty **then return** failure

**choose** a leaf node and remove it from the *fringe*

**if** the node contains a goal state **then return** the corresponding solution

add the node to the *explored*

expand the chosen node, adding the resulting nodes to the *fringe*

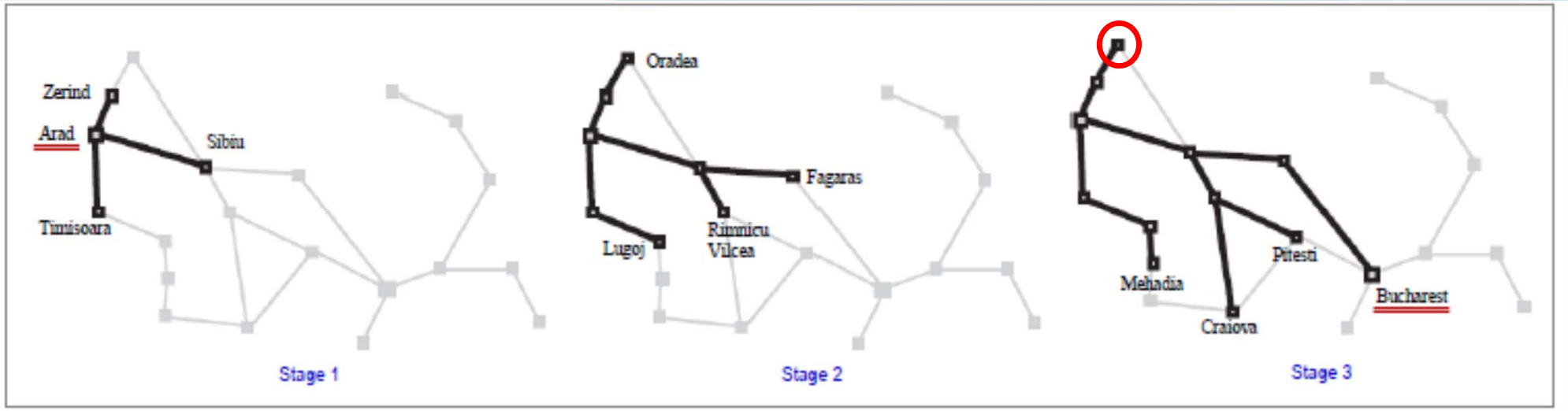
only if not in the *fringe* or *explored*

注：该 *explored*（亦称*closed list*）：一种数据结构，用于存储每个已扩展节点。

该算法中采用，将在*explored*或*fringe*中的曾出现过的节点丢弃的方法。

# 罗马尼亚问题的图搜索过程

通过图搜索在该罗马尼亚地图上生成一系列搜索路径。




每个路径在每个阶段 通过每一步加以扩展。
注意在第 3 阶段，最北部城市 (Oradea)的后继节点均已出现过，所以，相当于成为一个dead end.