南京大学
NANJING UNIVERSITY

人工智能导论

无信息搜索

郭兰哲

南京大学 智能科学与技术学院

Homepage: www.lamda.nju.edu.cn/guolz

Email: guolz@nju.edu.cn

提纲

- 搜索问题
- 无信息搜索
  - 宽度优先搜索
  - 深度优先搜索
  - 迭代加深的深度优先搜索
  - 代价一致性搜索
- 本章小结

提纲

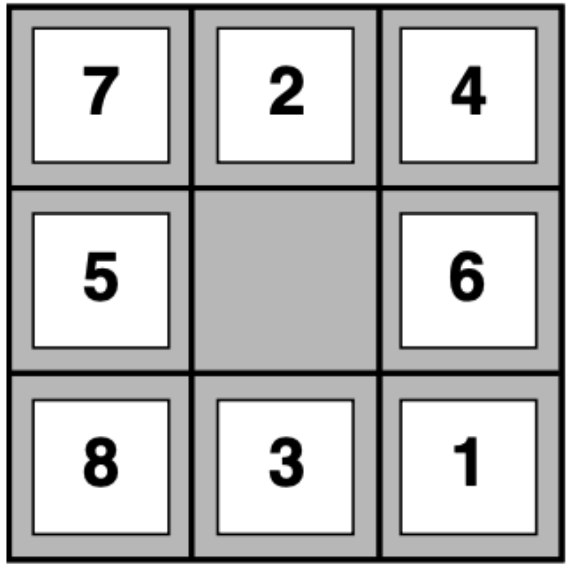
- 搜索问题
- 无信息搜索
  - 宽度优先搜索
  - 深度优先搜索
  - 迭代加深的深度优先搜索
  - 代价一致性搜索
- 本章小结

搜索问题

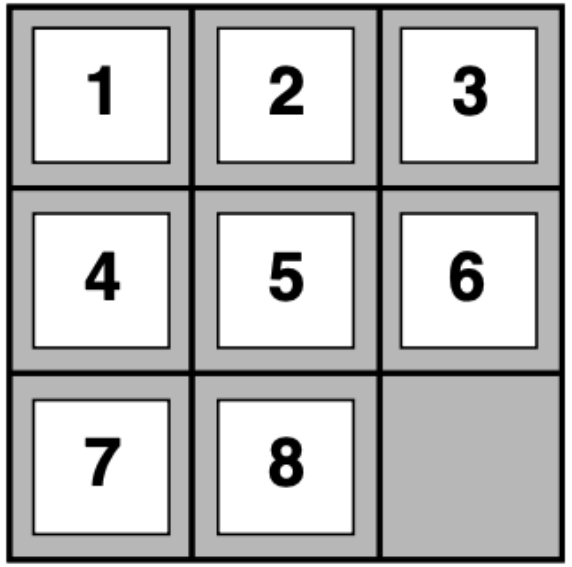
- 华容道

搜索问题

八数码



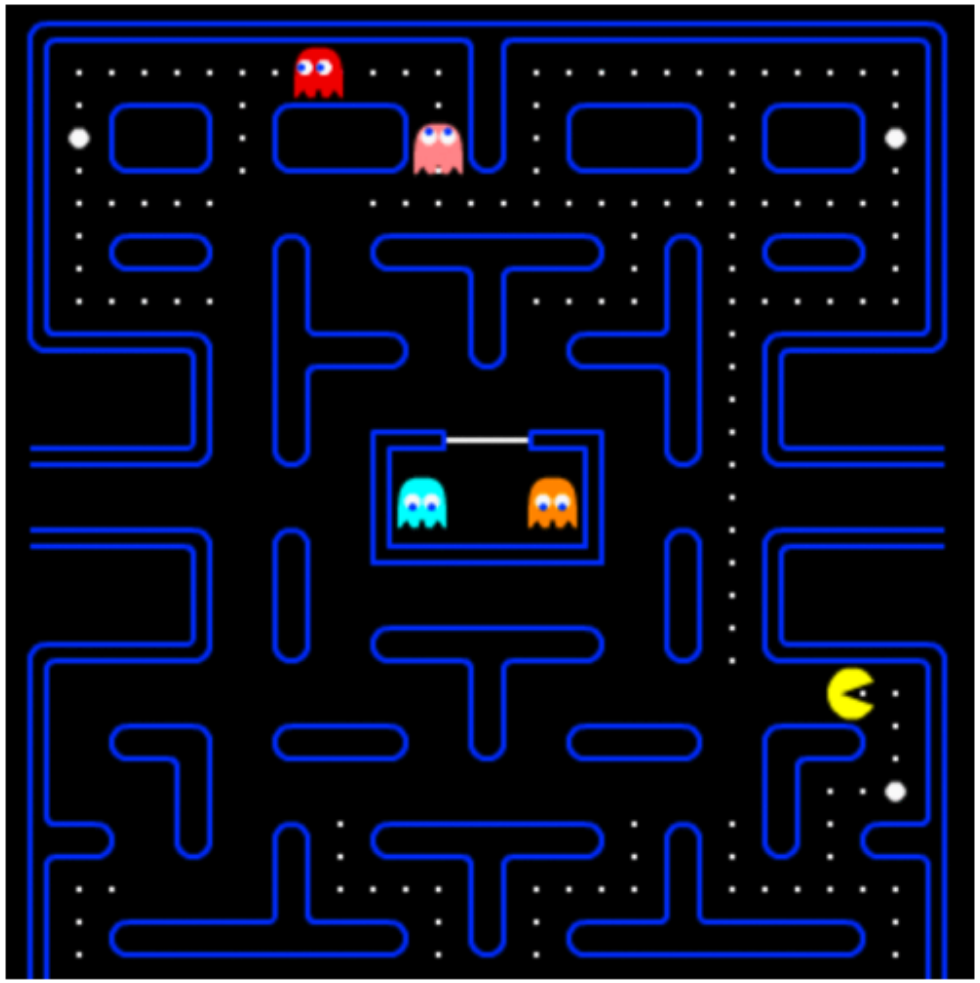
初始状态



目标状态

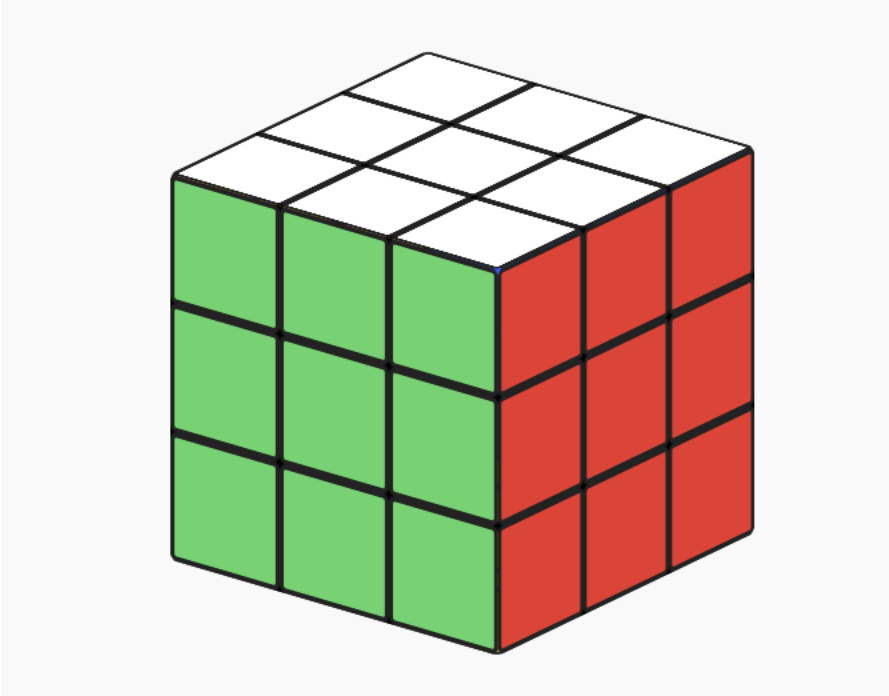
搜索问题

- 吃豆人



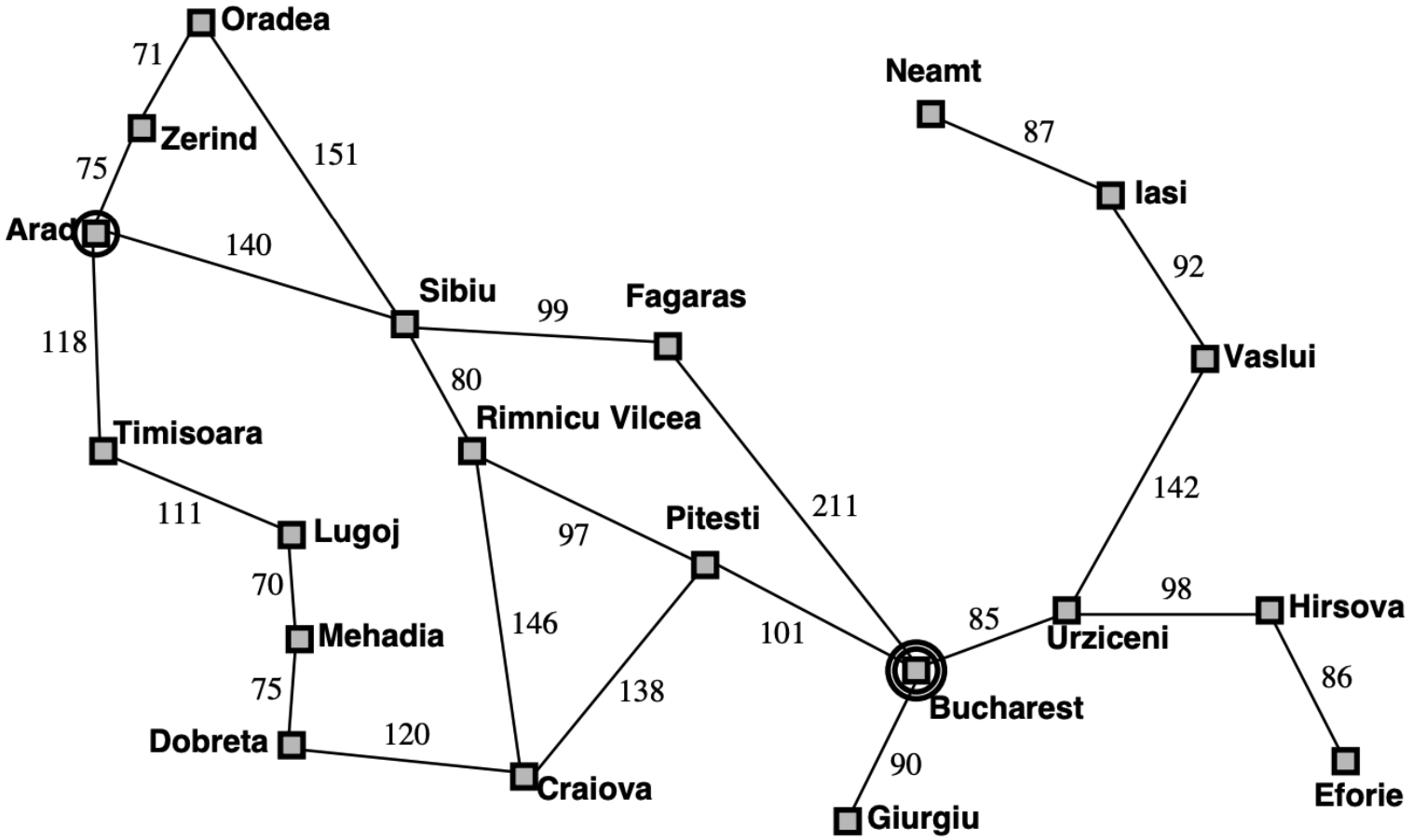
搜索问题

- 魔方



搜索问题

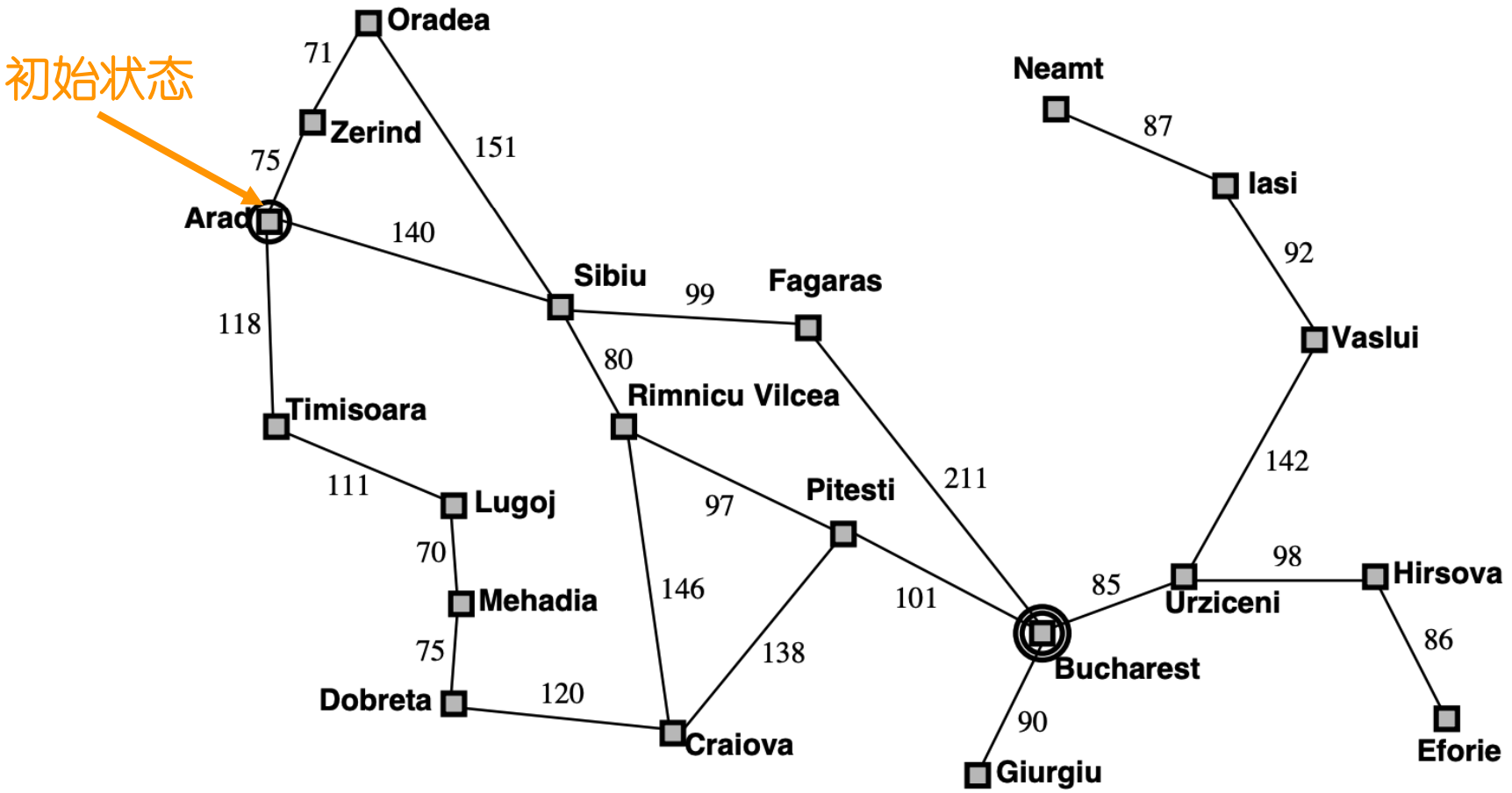
- 路径规划


Oradea
71
Neamt
87
Zerind
151
75
Iasi
Arad
140
92
Sibiu
99
Fagaras
118
Vaslui
80
Rimnicu Vilcea
Timisoara
142
111
Pitesti
211
Lugoj
97
98
70
Hirsova
Mehadia
146
85
101
Urziceni
75
138
86
Bucharest
Dobreta
120
90
Craiova
Eforie
Giurgiu

搜索问题

一个搜索问题可以用5个组成部分形式化描述

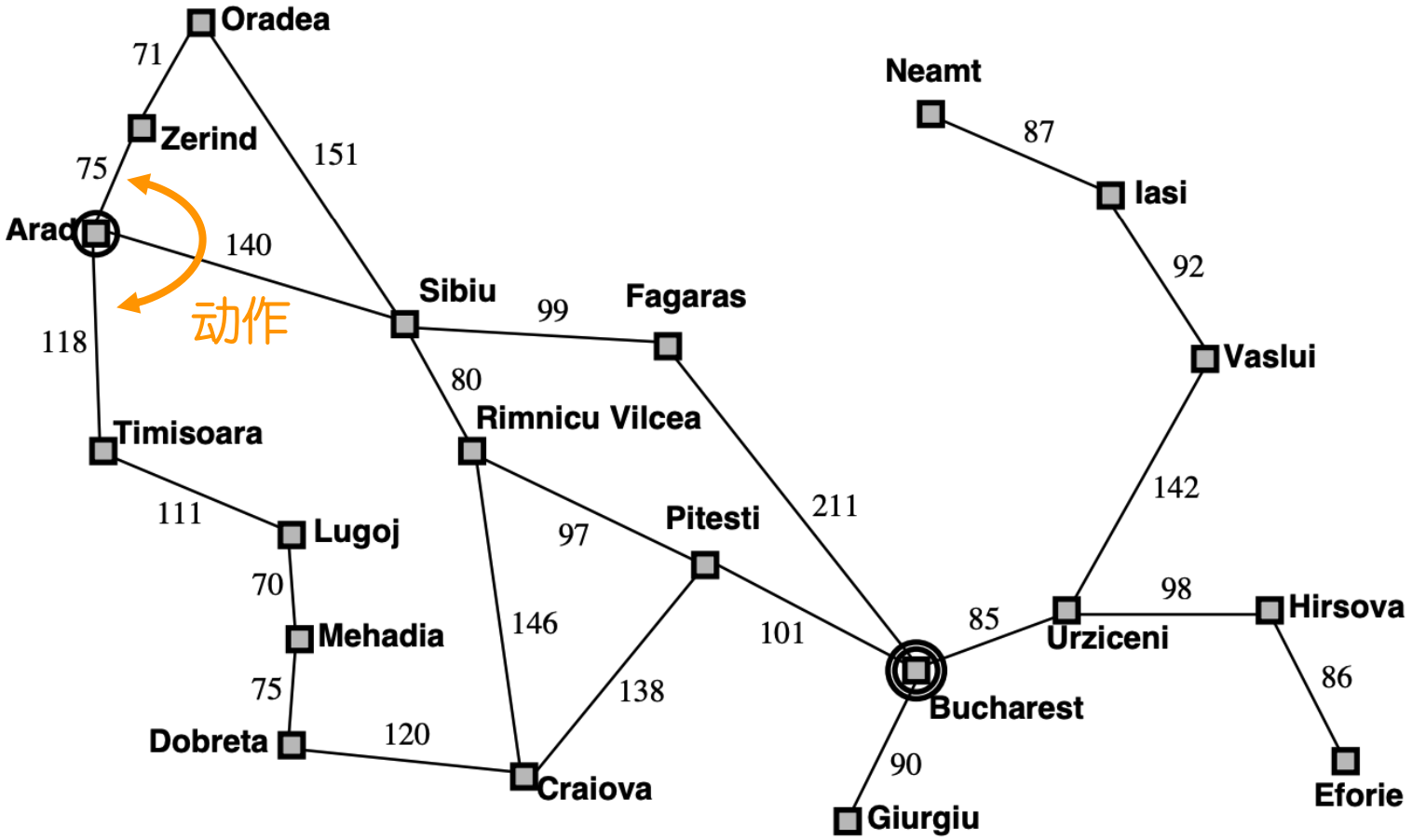
- 状态空间
- 初始状态
- 动作
- 状态转移
- 目标测试
- 路径/代价



搜索问题

一个搜索问题可以用5个组成部分形式化描述

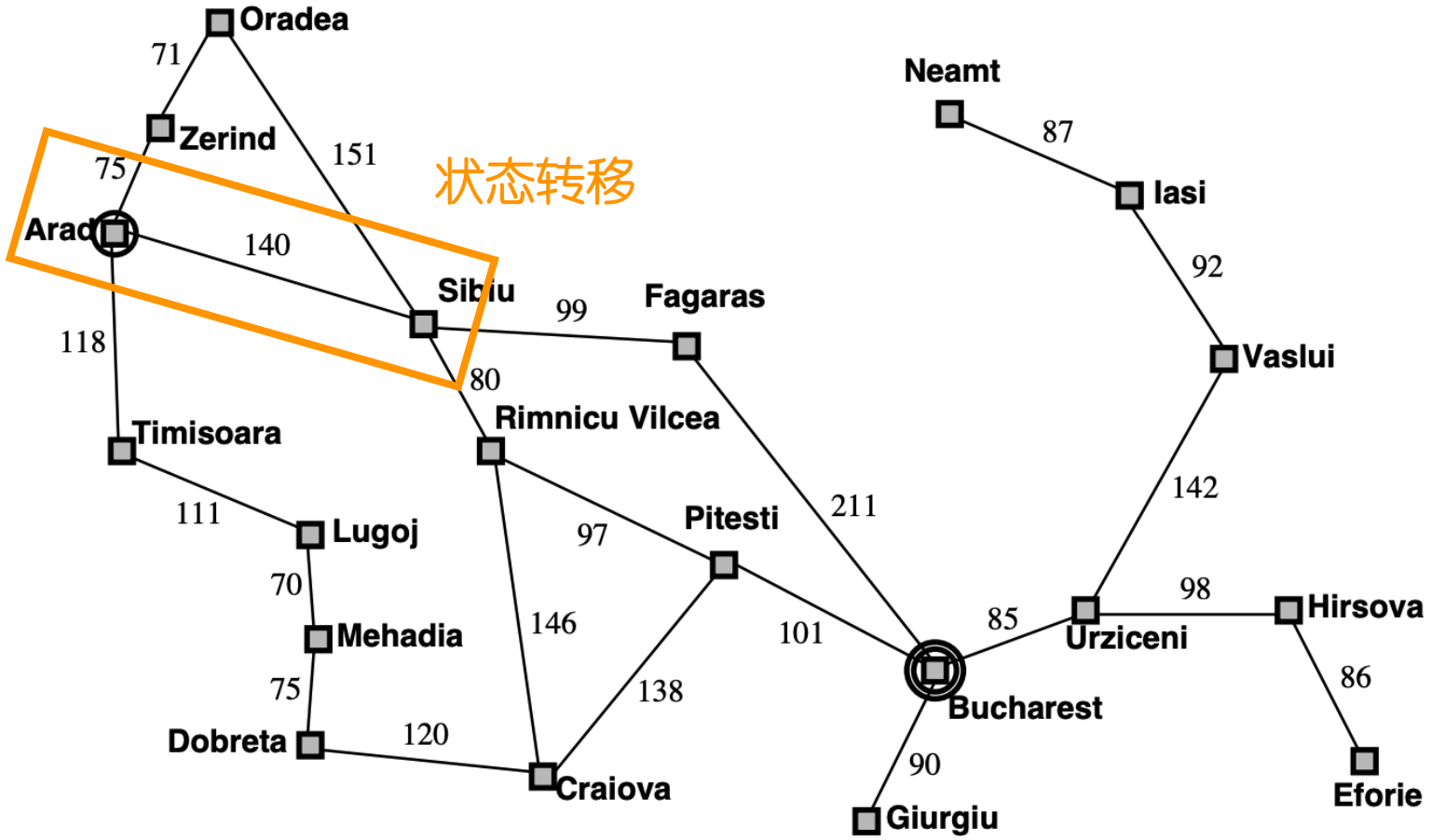
- 状态空间
- 初始状态
- 动作
- 状态转移
- 目标测试
- 路径/代价



搜索问题

一个搜索问题可以用5个组成部分形式化描述

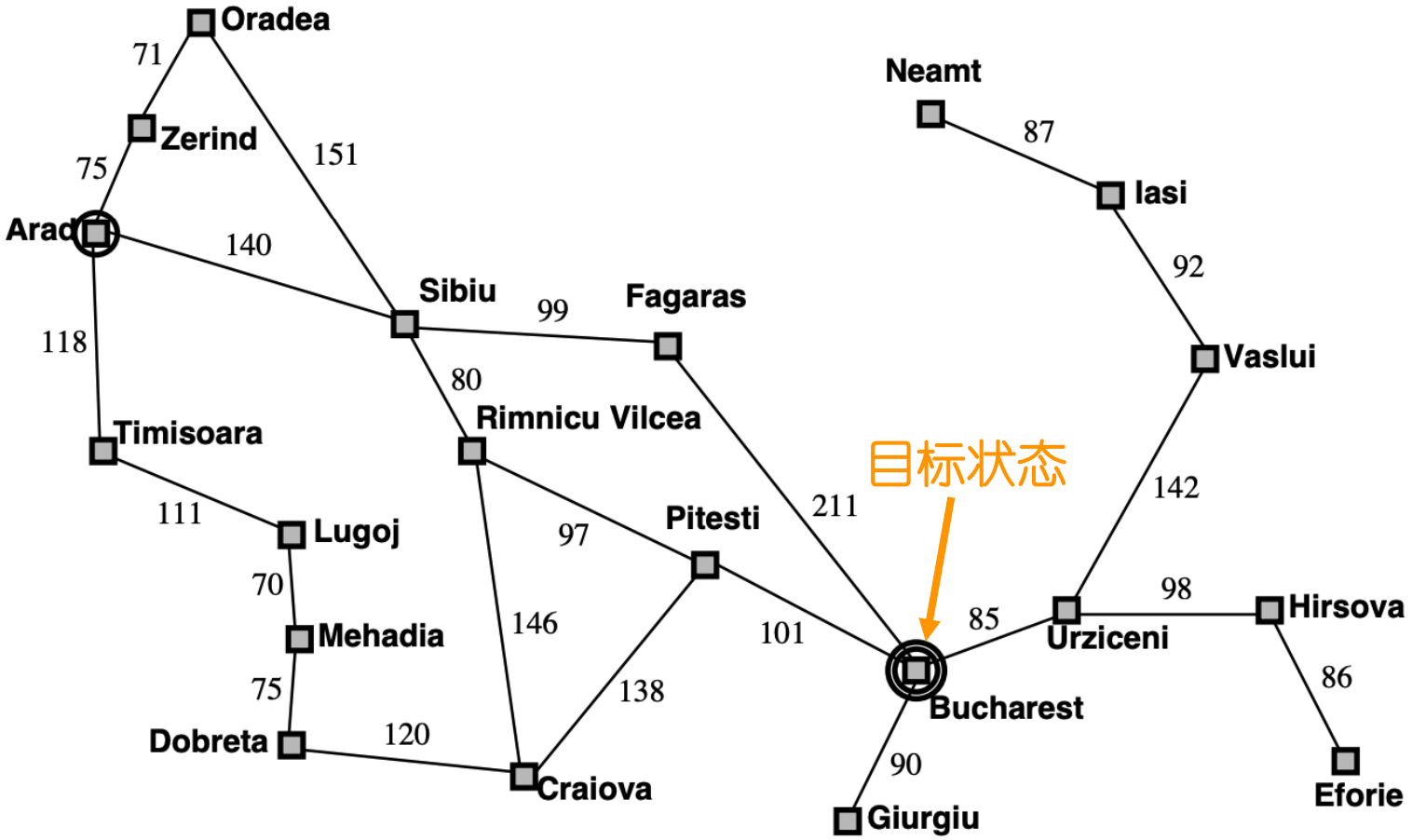
- 状态空间
- 初始状态
- 动作
- 状态转移
- 目标测试
- 路径/代价



搜索问题

一个搜索问题可以用5个组成部分形式化描述

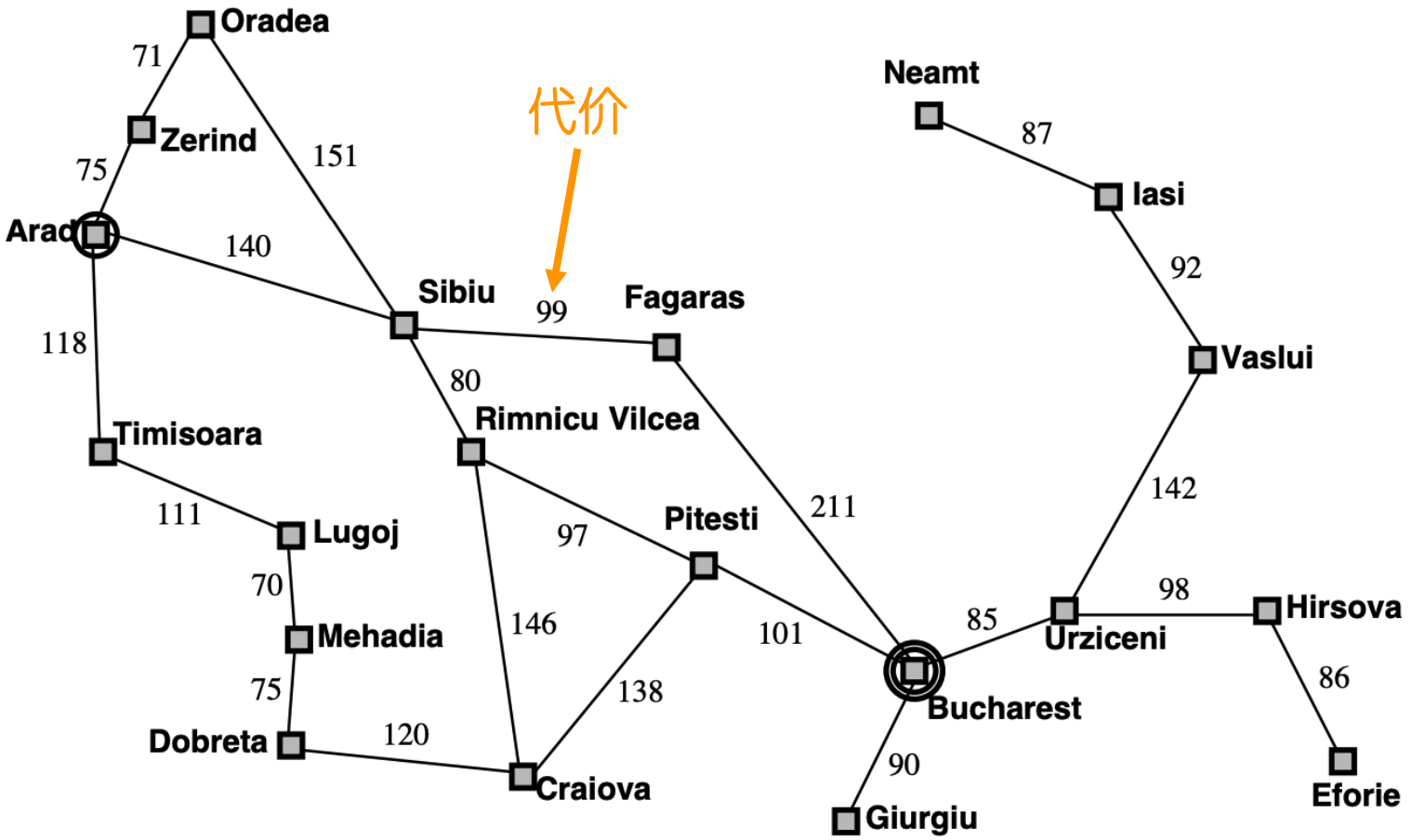
- 状态空间
- 初始状态
- 动作
- 状态转移
- 目标测试
- 路径/代价



搜索问题

一个搜索问题可以用5个组成部分形式化描述

- 状态空间
- 初始状态
- 动作
- 状态转移
- 目标测试
- 路径/代价



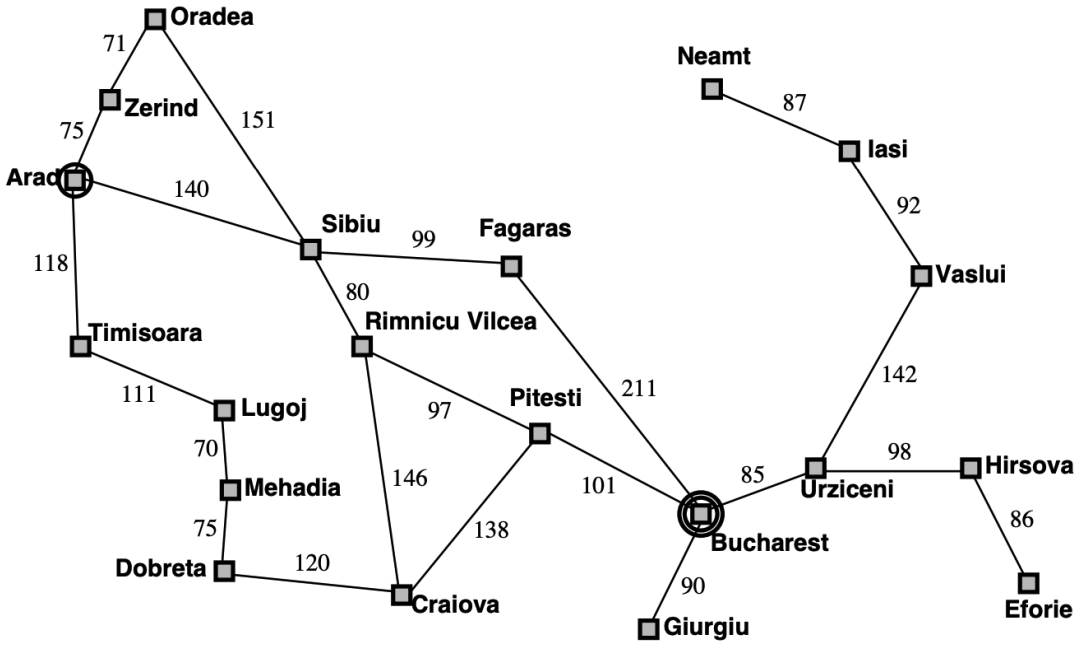
搜索问题

一个搜索问题可以用5个组成部分形式化描述

- 状态空间
- 初始状态
- 动作
- 状态转移
- 目标测试
- 路径/代价

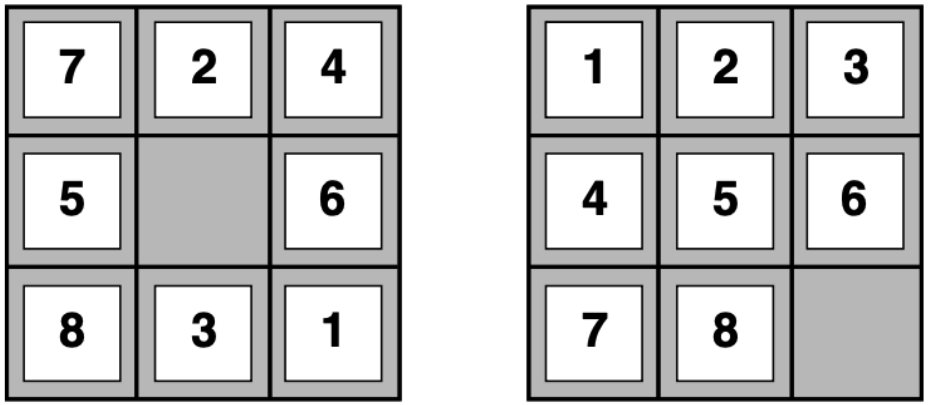
问题的解就是从初始状态到目标状态的一组行动序列

所有解里面代价最小的解即为最优解



搜索问题

八数码

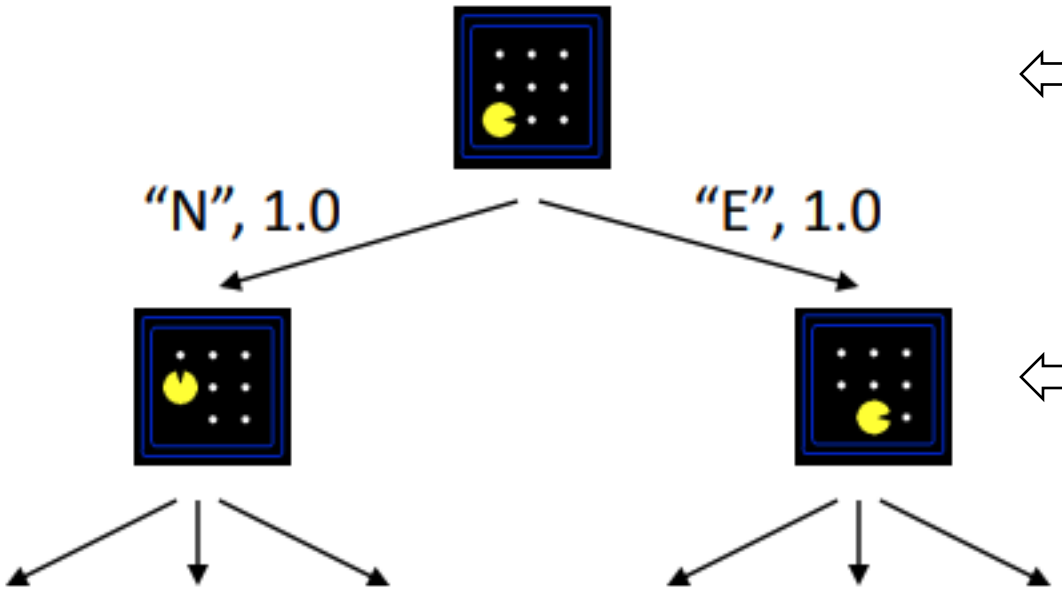


- 状态：8个数字方块和空格在棋盘上的分布
- 初始状态：任何状态都可能是初始状态
- 动作：移动空格
- 目标测试：检查是否到达目标状态
- 路径代价：每一步的代价为1

现实世界的搜索问题

- 游戏
- 旅行规划问题
- 旅行商问题
- 芯片布线问题
- 机器人导航问题
- 蛋白质设计问题
- ...

搜索树

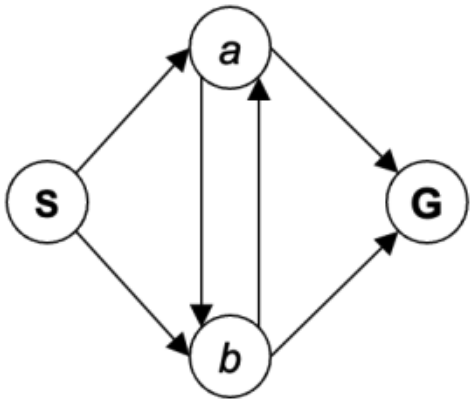


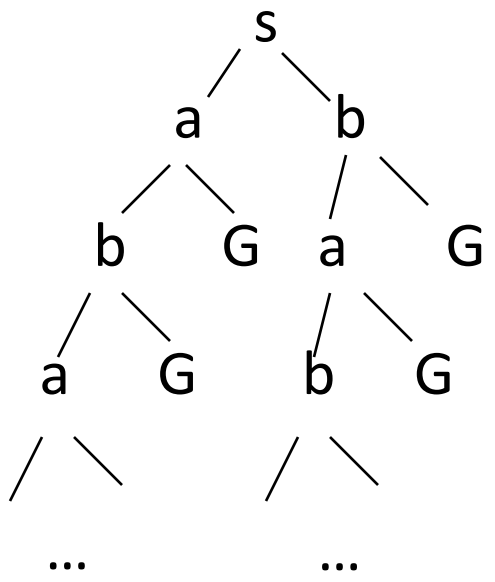
当前状态

可能的后继状态

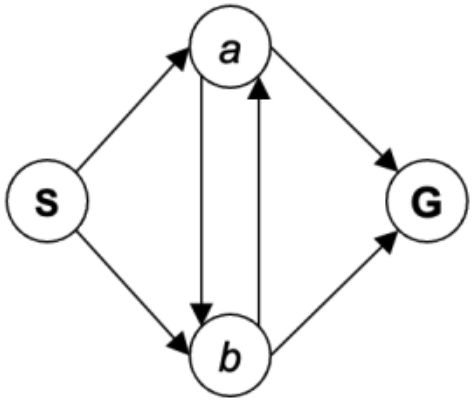
对于大多数问题，很难构建完整的搜索树

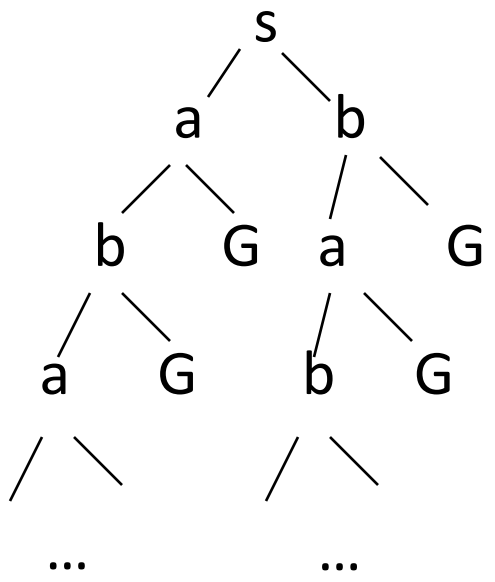
搜索树的深度



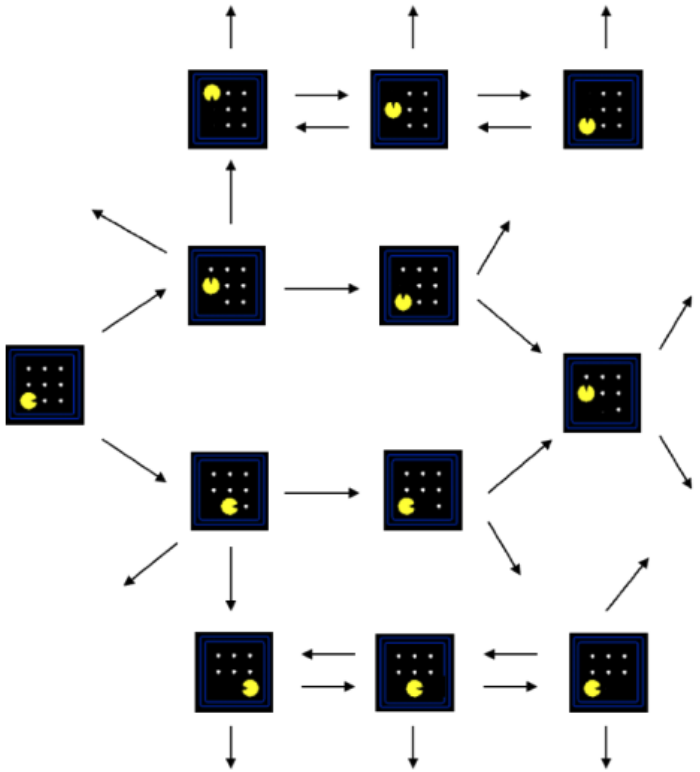


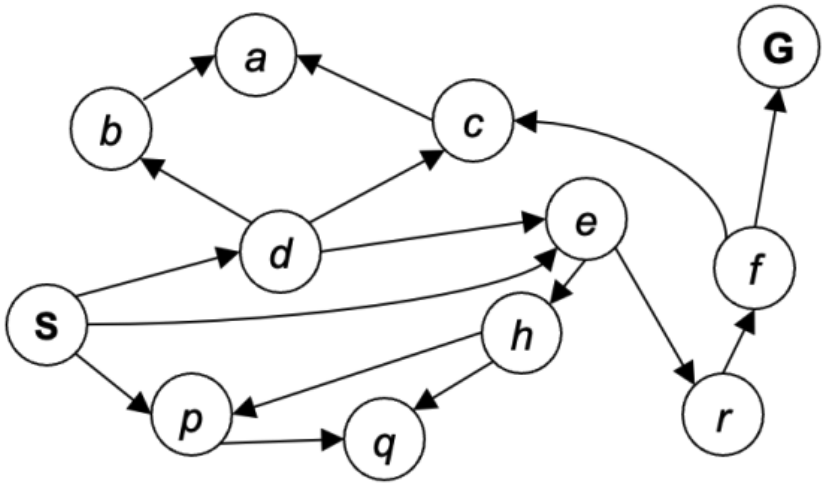
搜索树的深度





搜索图

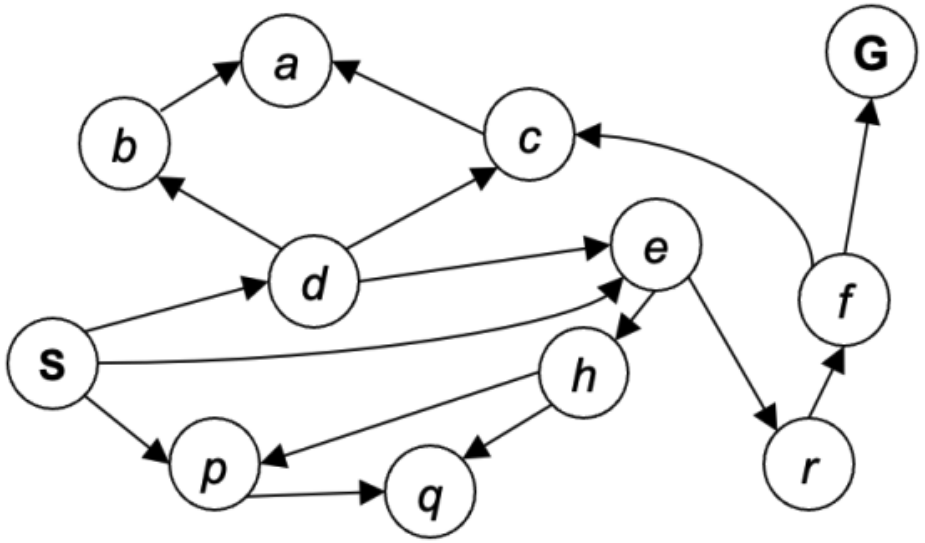


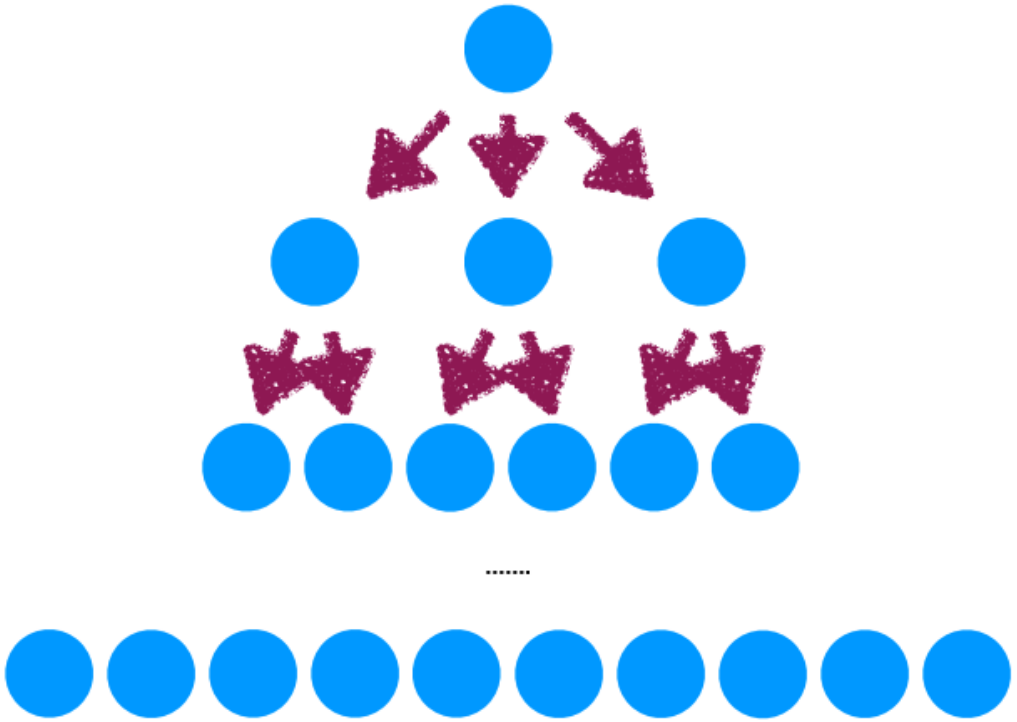


每个节点都是一个状态，图中的边体现了状态的转移

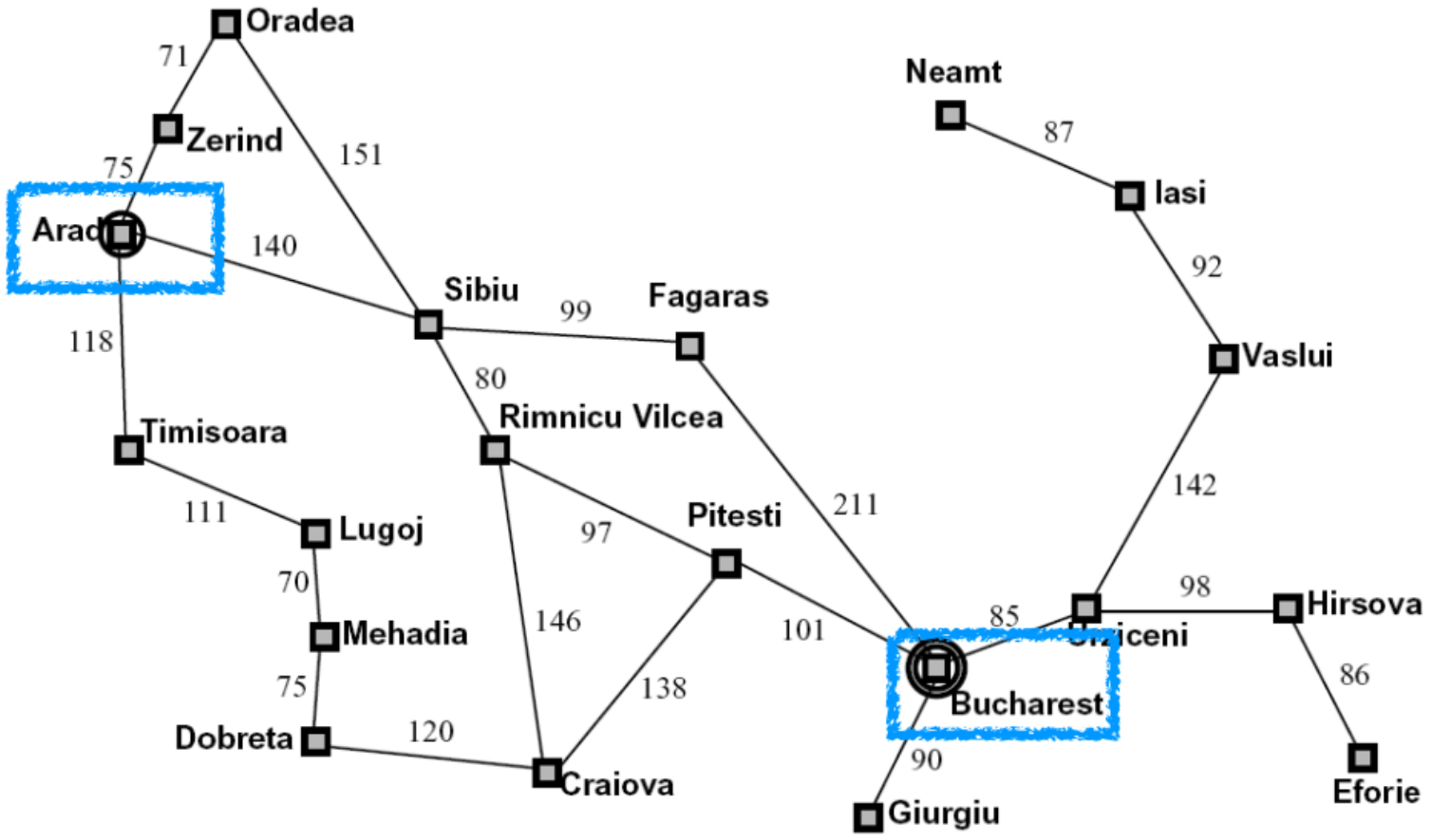
搜索图和搜索树

一个解是一个行动序列，搜索算法的工作就是考虑各种可能的行动序列

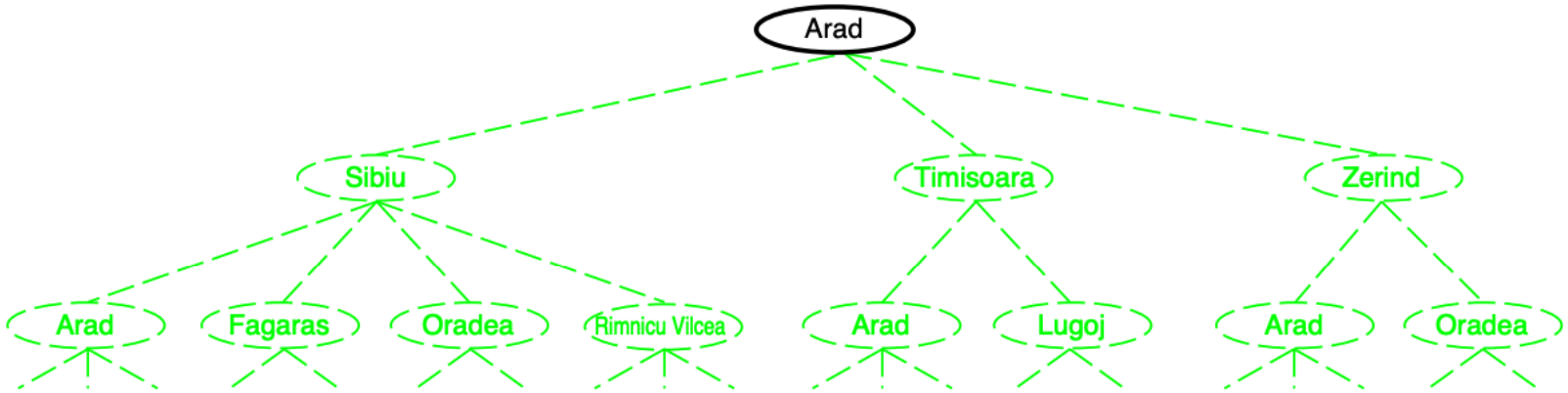




搜索问题求解

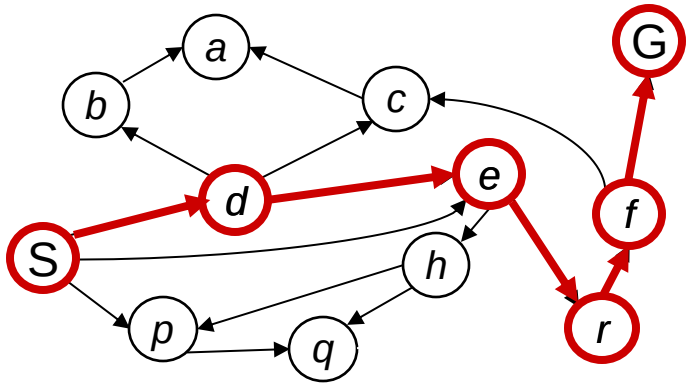

Oradea
71
Neamt
Zerind
87
151
75
Iasi
Arad
140
92
Sibiu
99
Fagaras
118
Vaslui
80
Rimnicu Vilcea
Timisoara
142
111
211
Pitesti
Lugoj
97
98
70
Hirsova
85
146
101
Mehadia
Urziceni
75
138
86
Bucharest
Dobreta
120
90
Craiova
Eforie
Giurgiu

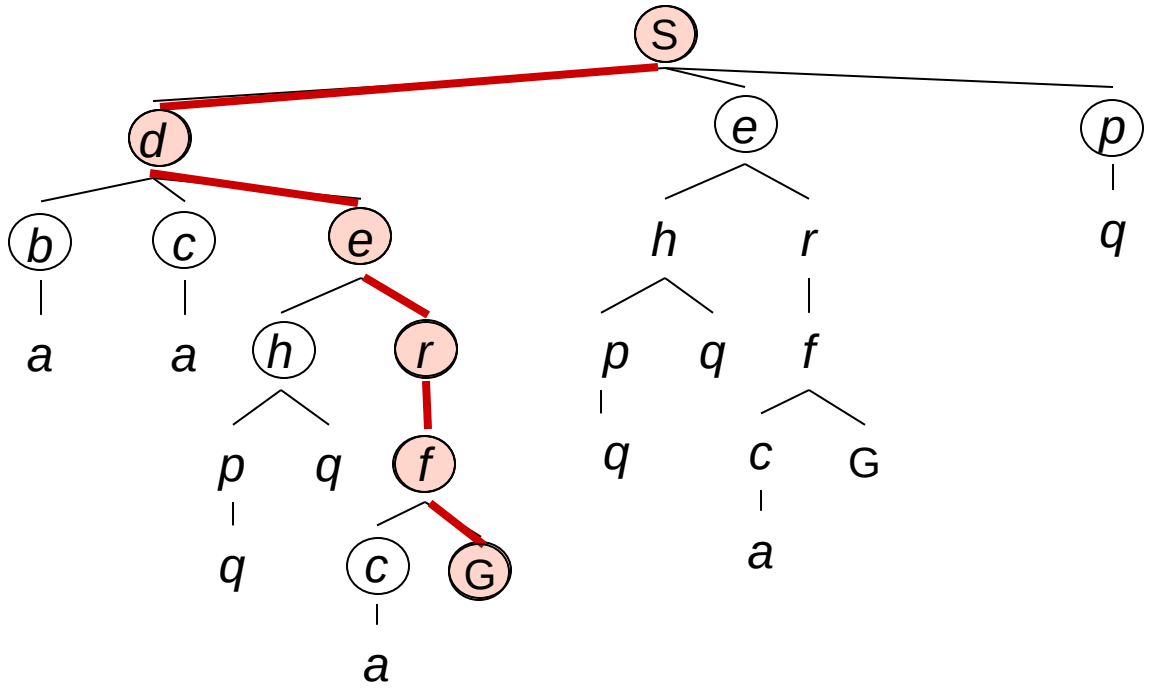
搜索问题求解



- 扩展当前的状态，在当前状态下采用各种行动，生成新的状态
- 所有待扩展的节点称为**边缘节点**
- 不断从边缘节点中选择节点并扩展，直到到达目标状态，或者没有状态可以扩展

树搜索算法





~~s~~

~~s → d~~

s → e

s → p

s → d → b

s → d → c

~~s → d → e~~

s → d → e → h

~~s → d → e → r~~

~~s → d → e → r → f~~

s → d → e → r → f → c

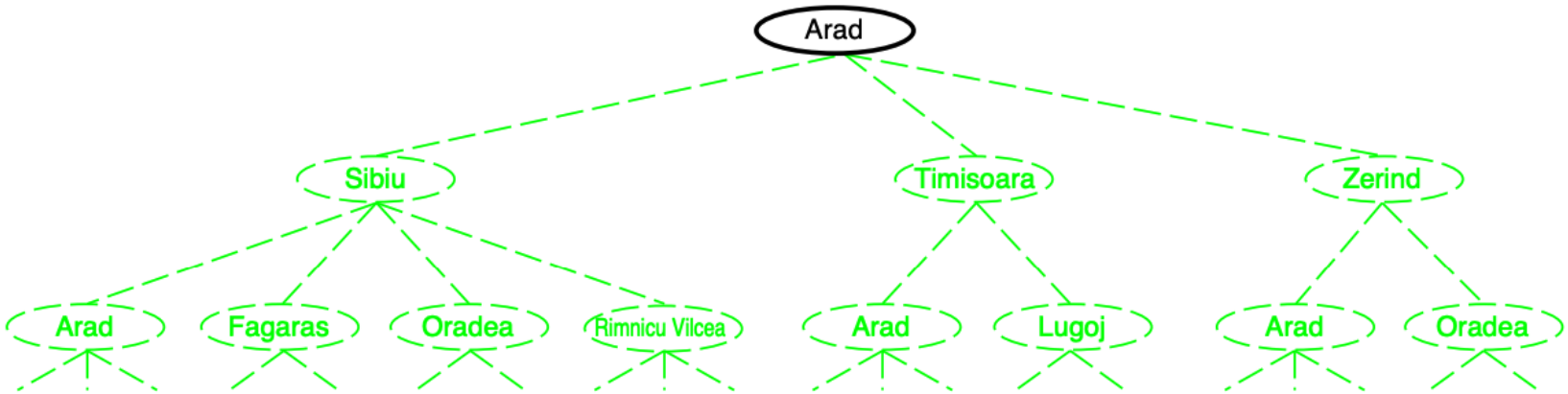
~~s → d → e → r → f → G~~

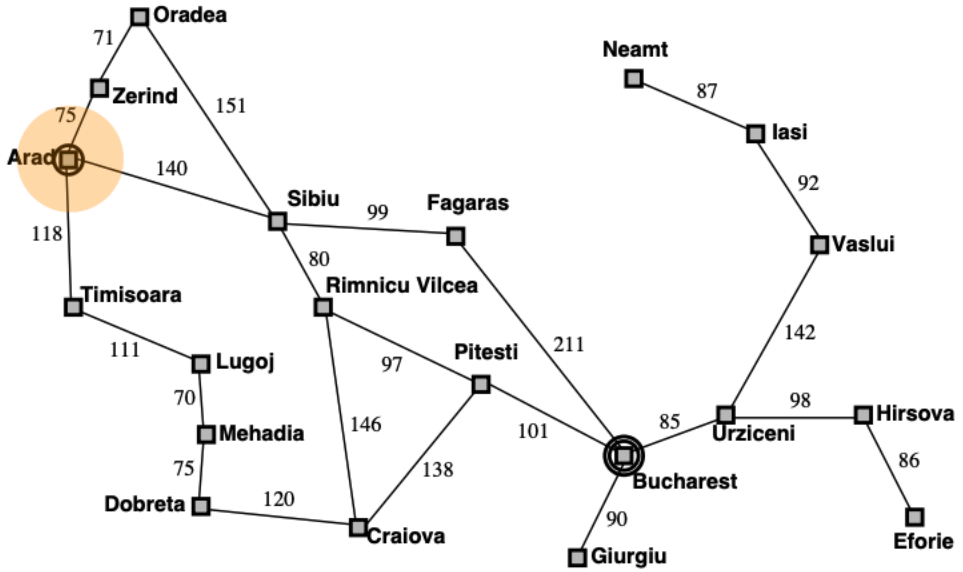
树搜索算法

```
function TREE-SEARCH( problem, strategy) returns a solution, or failure
    initialize the search tree using the initial state of problem
    loop do
        if there are no candidates for expansion then return failure
        choose a leaf node for expansion according to strategy
        if the node contains a goal state then return the corresponding solution
        else expand the node and add the resulting nodes to the search tree
    end
```

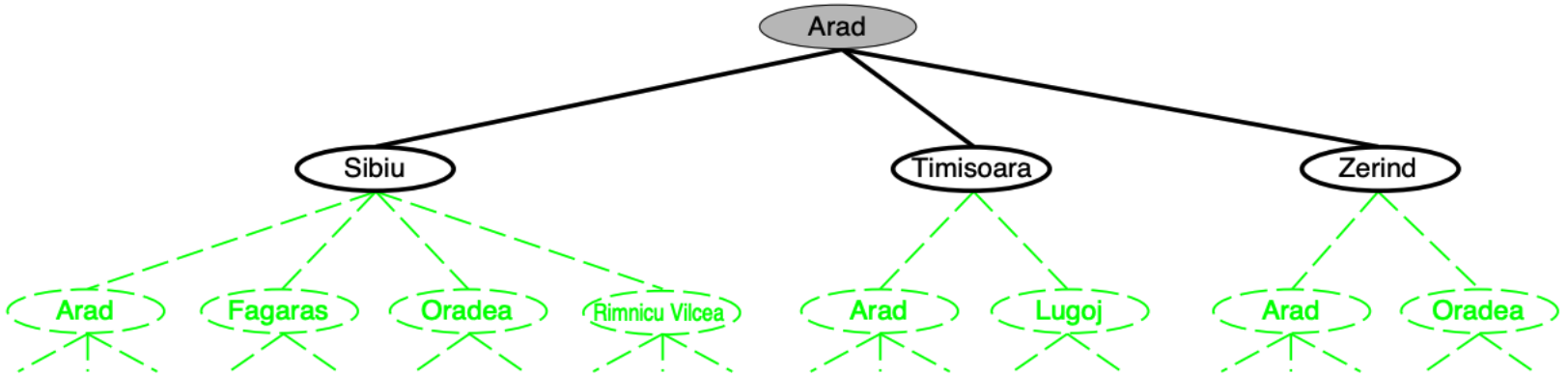
不同搜索策略的核心区别在于如何选择要扩展的边缘节点

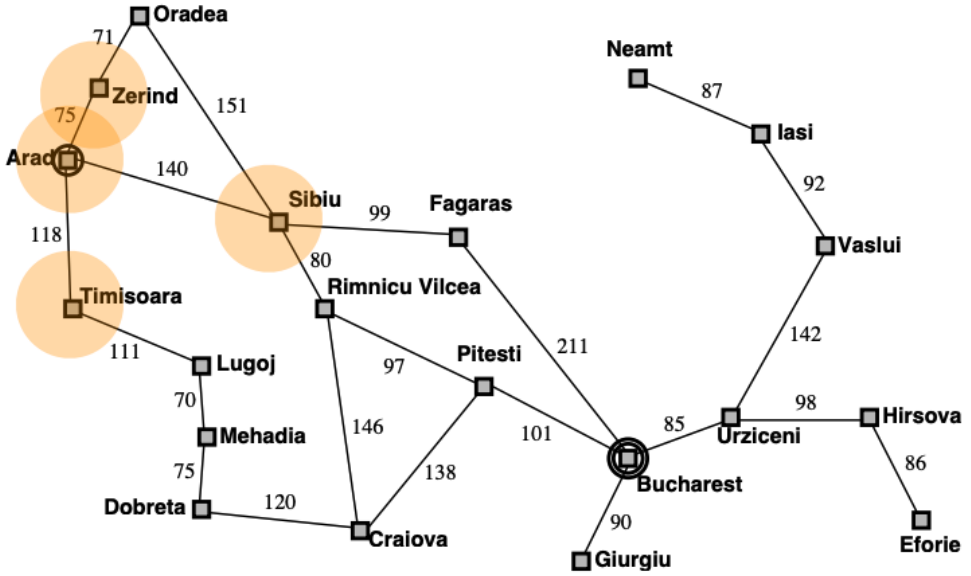
树搜索算法


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

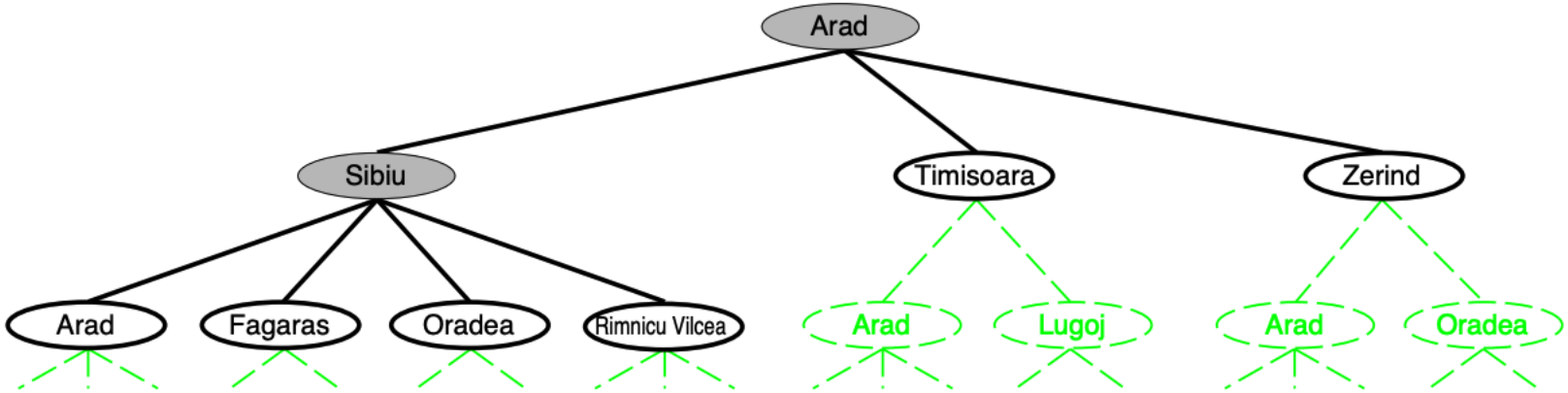

Oradea
71
Neamt
87
Zerind
151
Iasi
75
Arad
140
92
Sibiu
99
Fagaras
Vaslui
118
80
Rimnicu Vilcea
Timisoara
142
111
Lugoj
97
Pitesti
211
98
70
Hirsova
Mehadia
146
85
Urziceni
101
75
138
86
Bucharest
Dobreta
120
90
Craiova
Eforie
Giurgiu

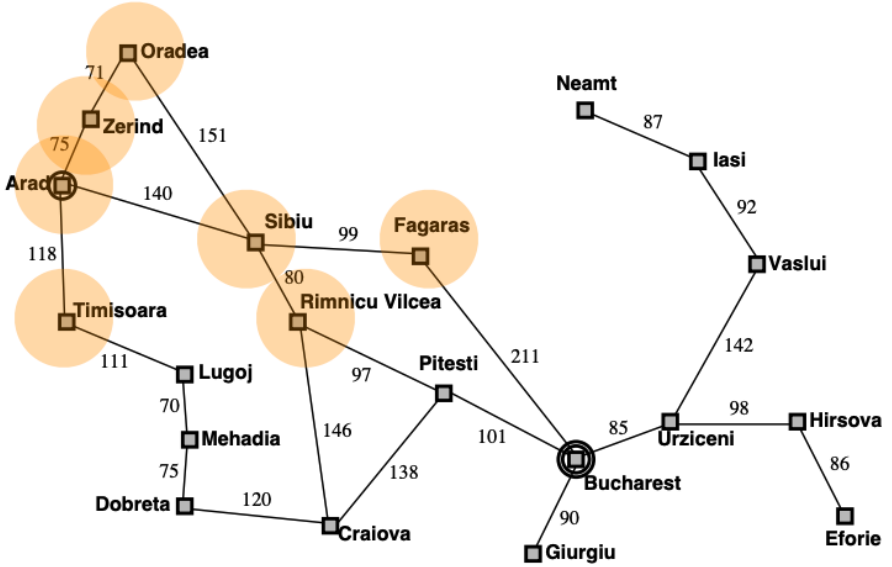
树搜索算法


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

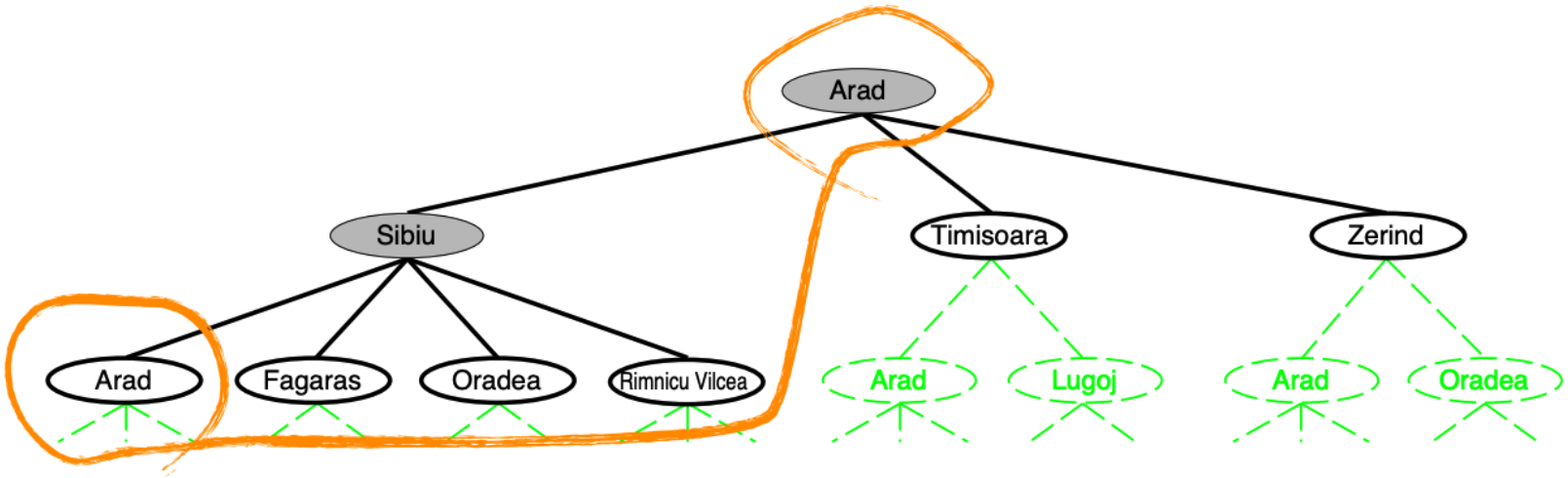

Oradea
71
Neamt
87
Zerind
151
75
Iasi
Arad
140
92
Sibiu
99
Fagaras
Vaslui
118
80
Rimnicu Vilcea
Timisoara
142
111
Lugoj
97
Pitesti
211
70
98
Hirsova
Mehadia
146
85
Urziceni
101
75
138
86
Bucharest
Dobreta
120
Craiova
90
Eforie
Giurgiu

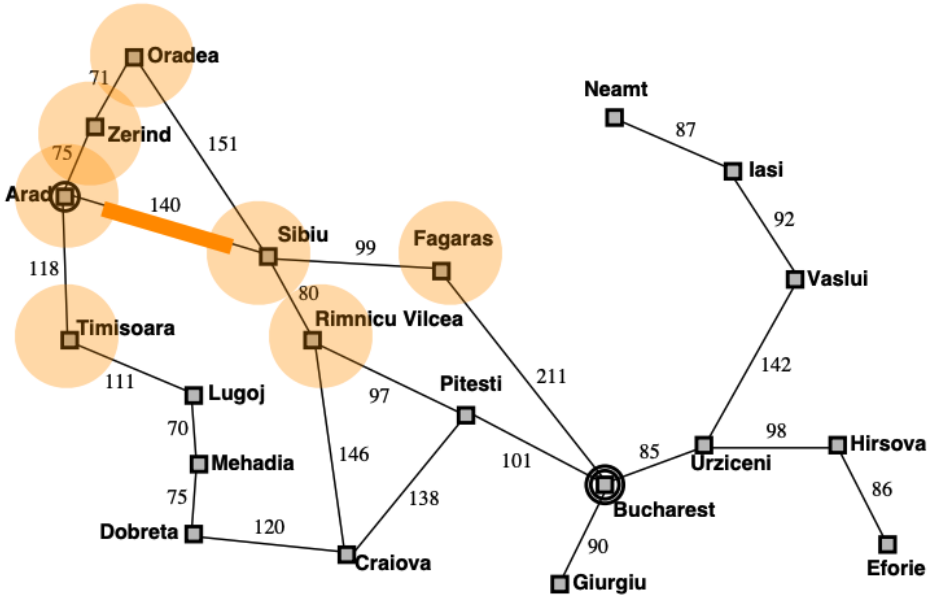
树搜索算法


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


Oradea
71
Neamt
87
Zerind
151
Iasi
75
92
Arad
140
Sibiu
99
Fagaras
Vaslui
118
80
Rimnicu Vilcea
Timisoara
142
111
Lugoj
97
Pitesti
211
70
98
Hirsova
Mehadia
146
85
Urziceni
101
75
138
Bucharest
86
Dobreta
120
Craiova
90
Giurgiu
Eforie

树搜索算法


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


Oradea
71
Neamt
87
Zerind
151
Iasi
75
Arad
140
Sibiu
99
Fagaras
92
Vaslui
118
80
Rimnicu Vilcea
Timisoara
142
111
Lugoj
97
Pitesti
211
70
98
Hirsova
Mehadia
146
85
Urziceni
101
75
138
Bucharest
86
Dobreta
120
Craiova
90
Eforie
Giurgiu

图搜索算法

```
function TREE-SEARCH( problem, strategy) returns a solution, or failure
    initialize the search tree using the initial state of problem
    loop do
        if there are no candidates for expansion then return failure
        choose a leaf node for expansion according to strategy
        if the node contains a goal state then return the corresponding solution
        else expand the node and add the resulting nodes to the search tree
    end
```

```
function GRAPH-SEARCH( problem, fringe) returns a solution, or failure
    closed ← an empty set
    fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
    loop do
        if fringe is empty then return failure
        node ← REMOVE-FRONT(fringe)
        if GOAL-TEST(problem, STATE[node]) then return node
        if STATE[node] is not in closed then
            add STATE[node] to closed
            fringe ← INSERTALL(EXPAND(node, problem), fringe)
    end
```

搜索策略

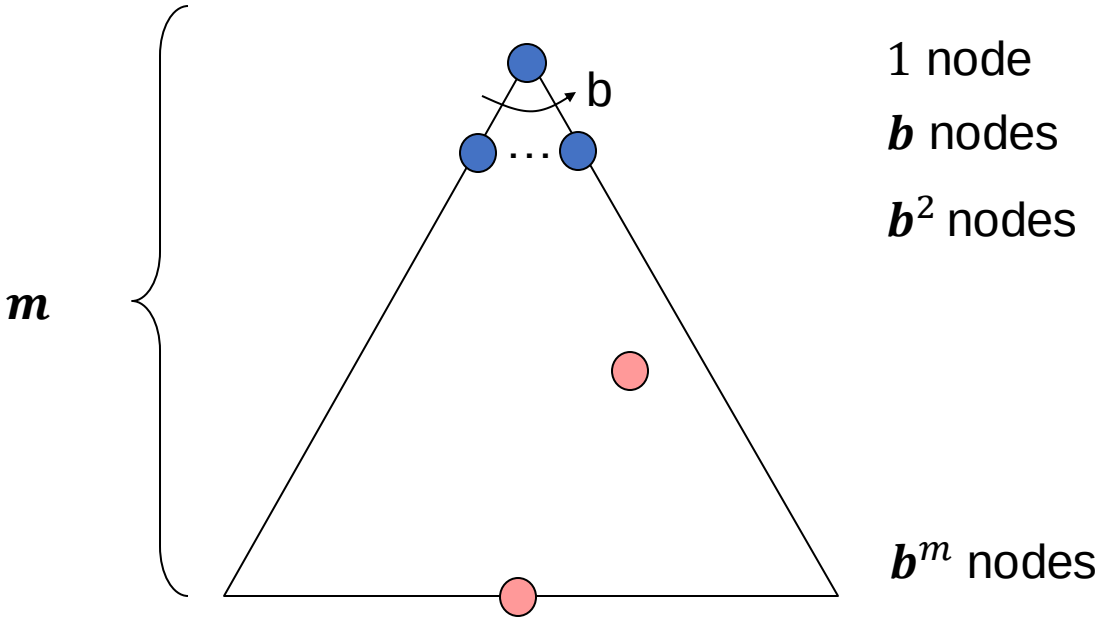
搜索策略的主要区别在于**如何选择要扩展的状态**

如何评估搜索算法的性能？

- ✓ **完备性**：当问题有解时，这个算法是否能保证找到解？
- ✓ **最优性**：搜索策略是否能找到最优解？
- ✓ **时间复杂度**：找到解需要花费多长时间？
- ✓ **空间复杂度**：在执行搜索的过程总需要多少内存？

搜索策略性能分析

符号	含义
b	分支因子，即搜索树中每个节点最大的分支数目
d	根节点到最浅的目标结点的路径长度
m	搜索树中路径的最大可能长度
n	状态空间中状态的数量



$$1 + b + b^2 + \cdots + b^m = O(b^m)$$

提纲

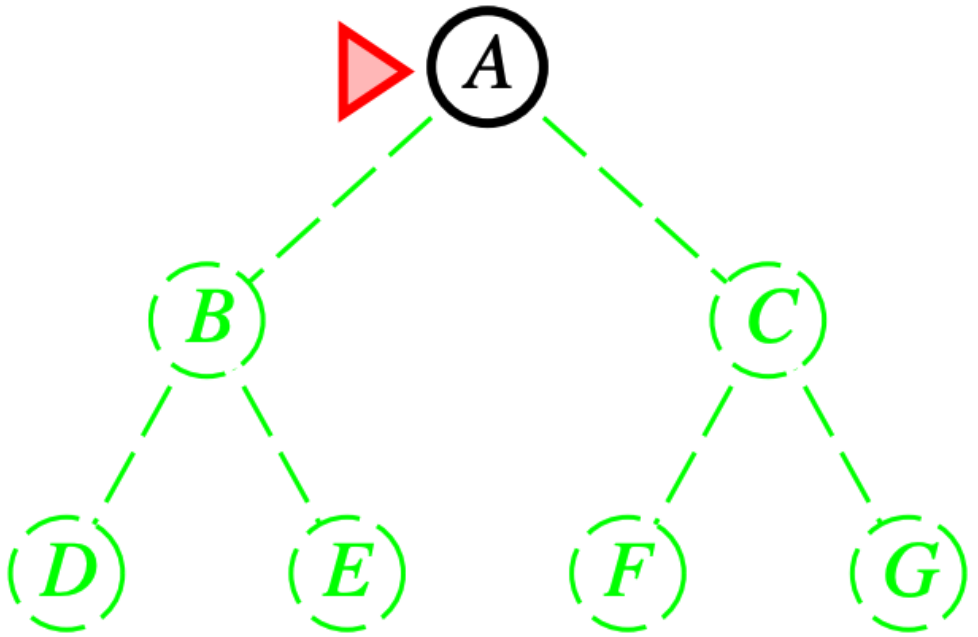
- 搜索问题
- 无信息搜索
  - 宽度优先搜索
  - 深度优先搜索
  - 迭代加深的深度优先搜索
  - 代价一致性搜索
- 本章小结

除了问题定义中给定的状态信息外没有任何附加信息

宽度优先搜索（breadth-first search, BFS）

先扩展根节点，接着扩展根节点的所有后继节点，然后再扩展它们的后继，以此类推。

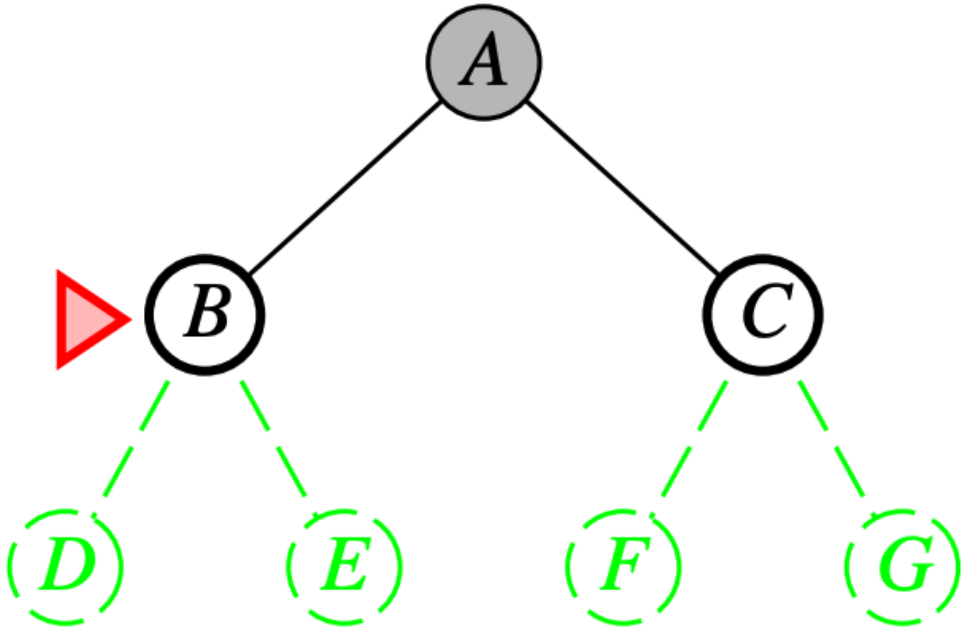
一般地，在下一层的任何节点扩展之前，搜索树本层深度的搜有节点应该都已经扩展过



宽度优先搜索（breadth-first search, BFS）

先扩展根节点，接着扩展根节点的所有后继节点，然后再扩展它们的后继，以此类推。

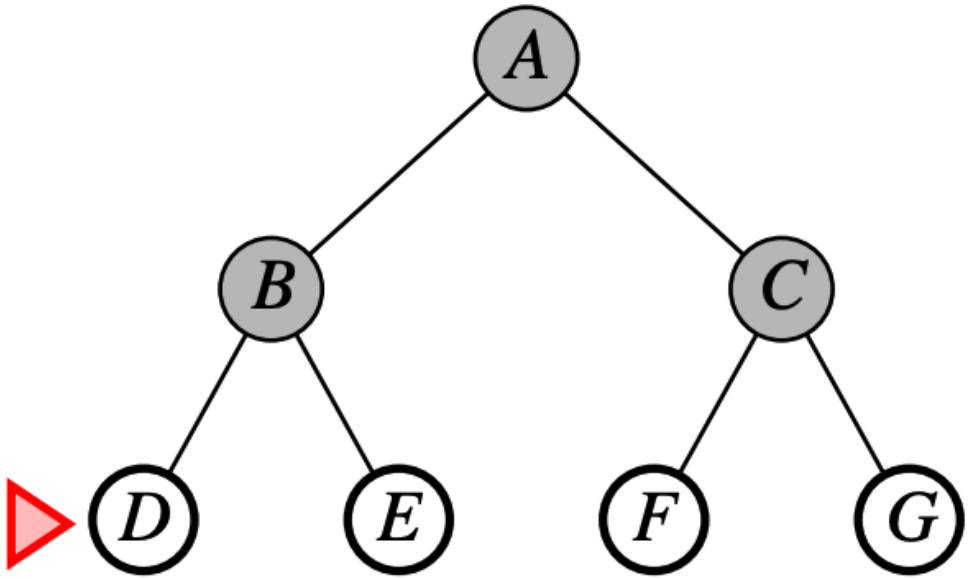
一般地，在下一层的任何节点扩展之前，搜索树本层深度的搜有节点应该都已经扩展过。



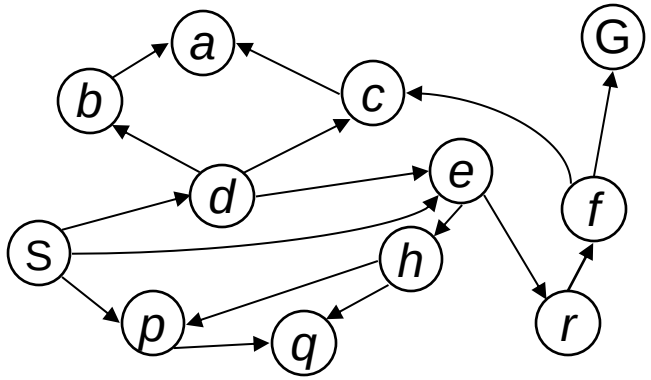
宽度优先搜索（breadth-first search, BFS）

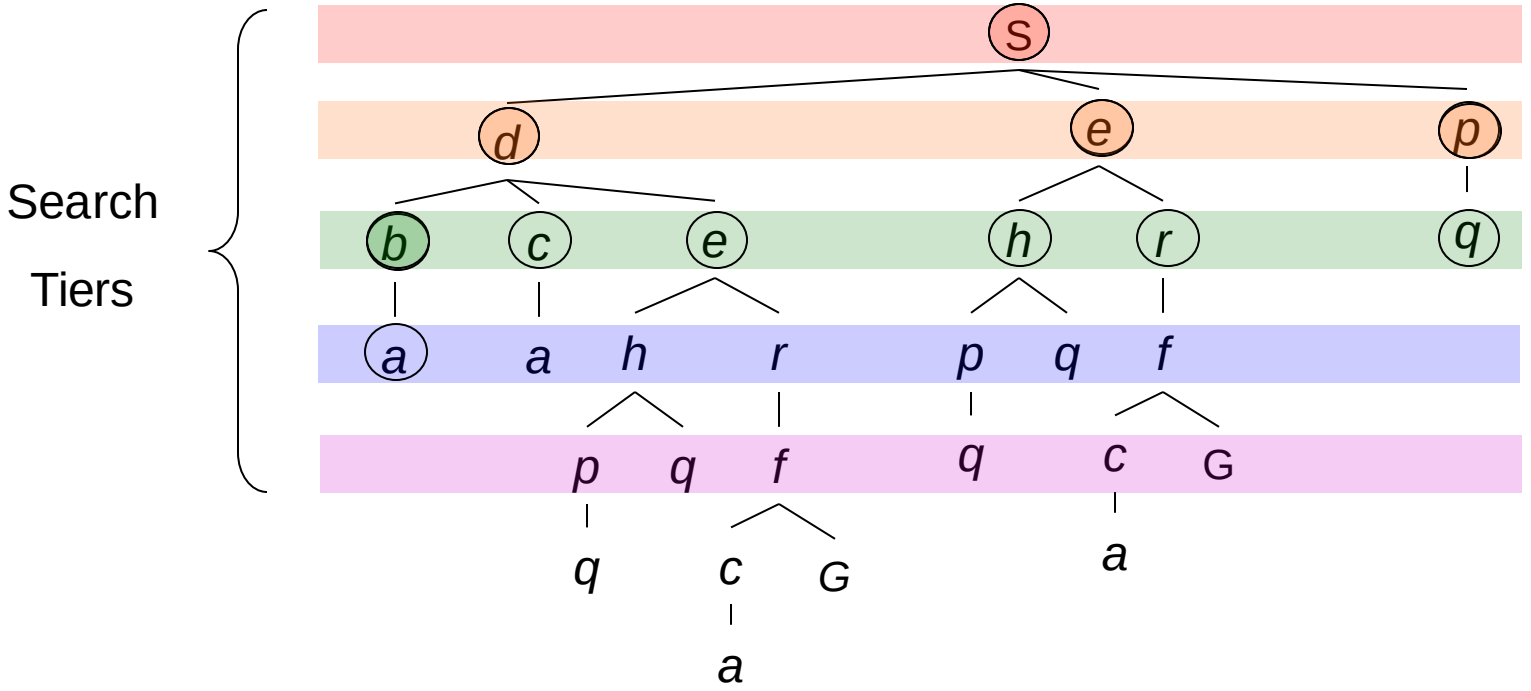
先扩展根节点，接着扩展根节点的所有后继节点，然后再扩展它们的后继，以此类推。

一般地，在下一层的任何节点扩展之前，搜索树本层深度的搜有节点应该都已经扩展过。



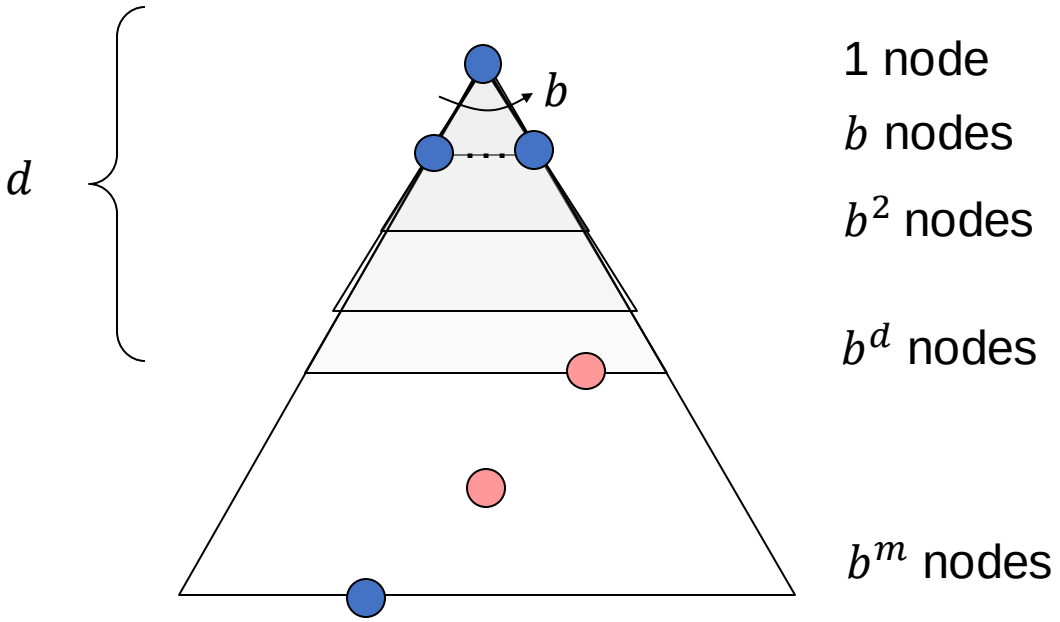
宽度优先搜索（breadth-first search, BFS）





性能分析

- 时间复杂度?
  - $1 + b + b^2 + b^3 + \cdots + b^d = O(b^d)$
- 空间复杂度?
  - $O(b^d)$
- 完备性?
  - Yes
- 最优性?
  - Yes， 如果所有行动代价相同

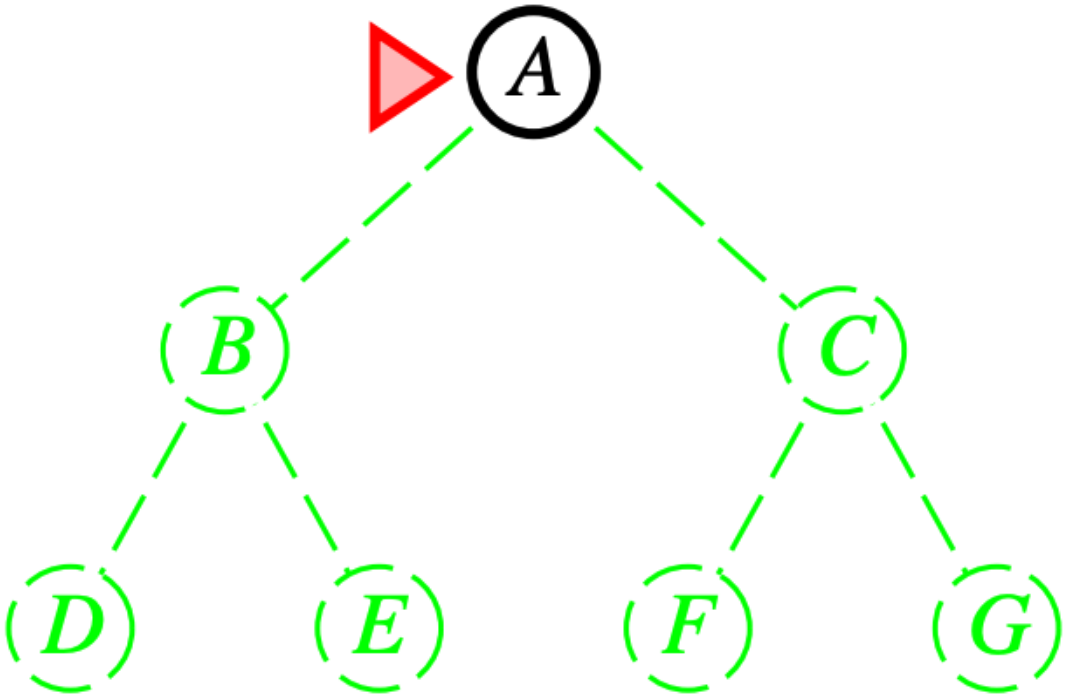


提纲

- 搜索问题
- 无信息搜索
  - 宽度优先搜索
  - 深度优先搜索
  - 迭代加深的深度优先搜索
  - 代价一致性搜索
- 本章小结

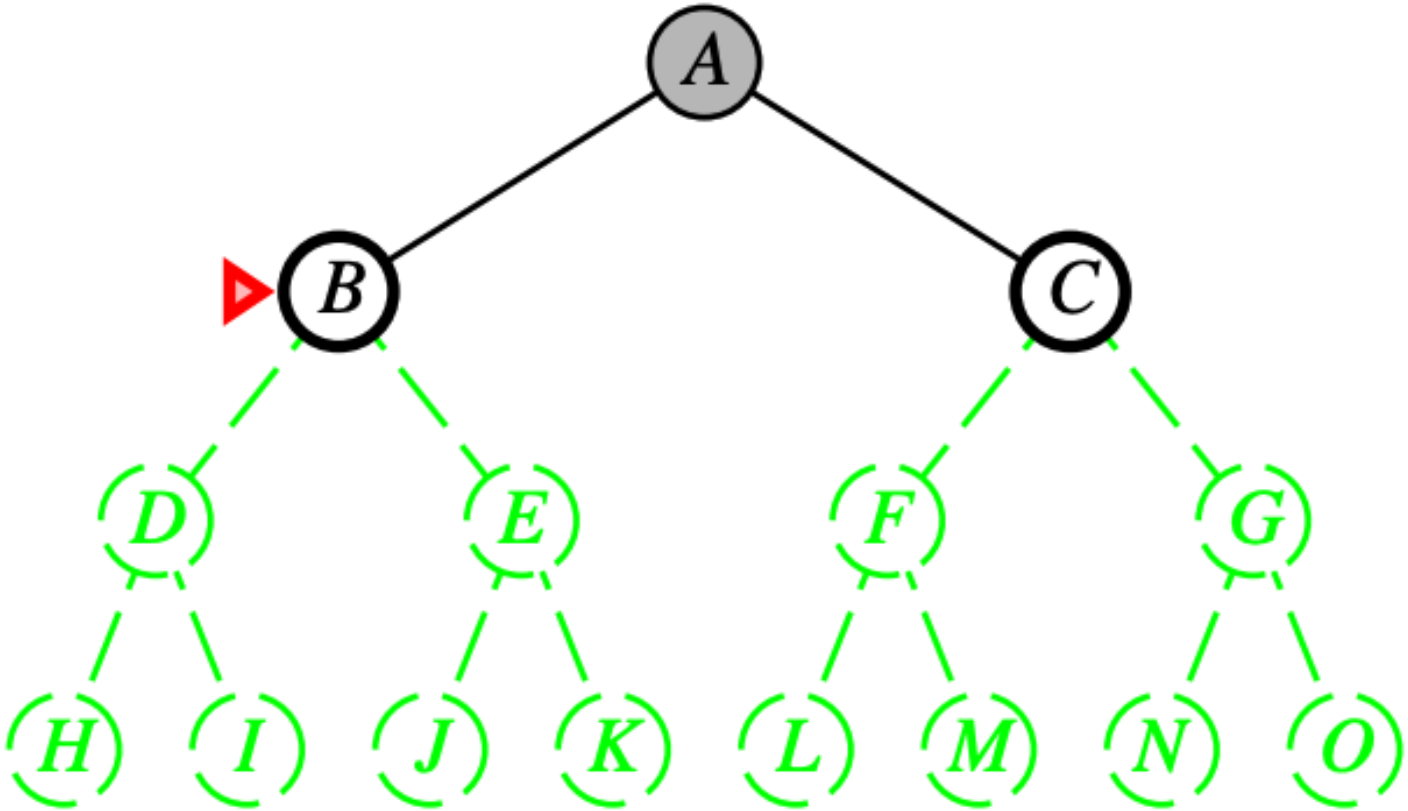
深度优先搜索（depth-first search, DFS）

总是扩展当前边缘节点集中最深的节点



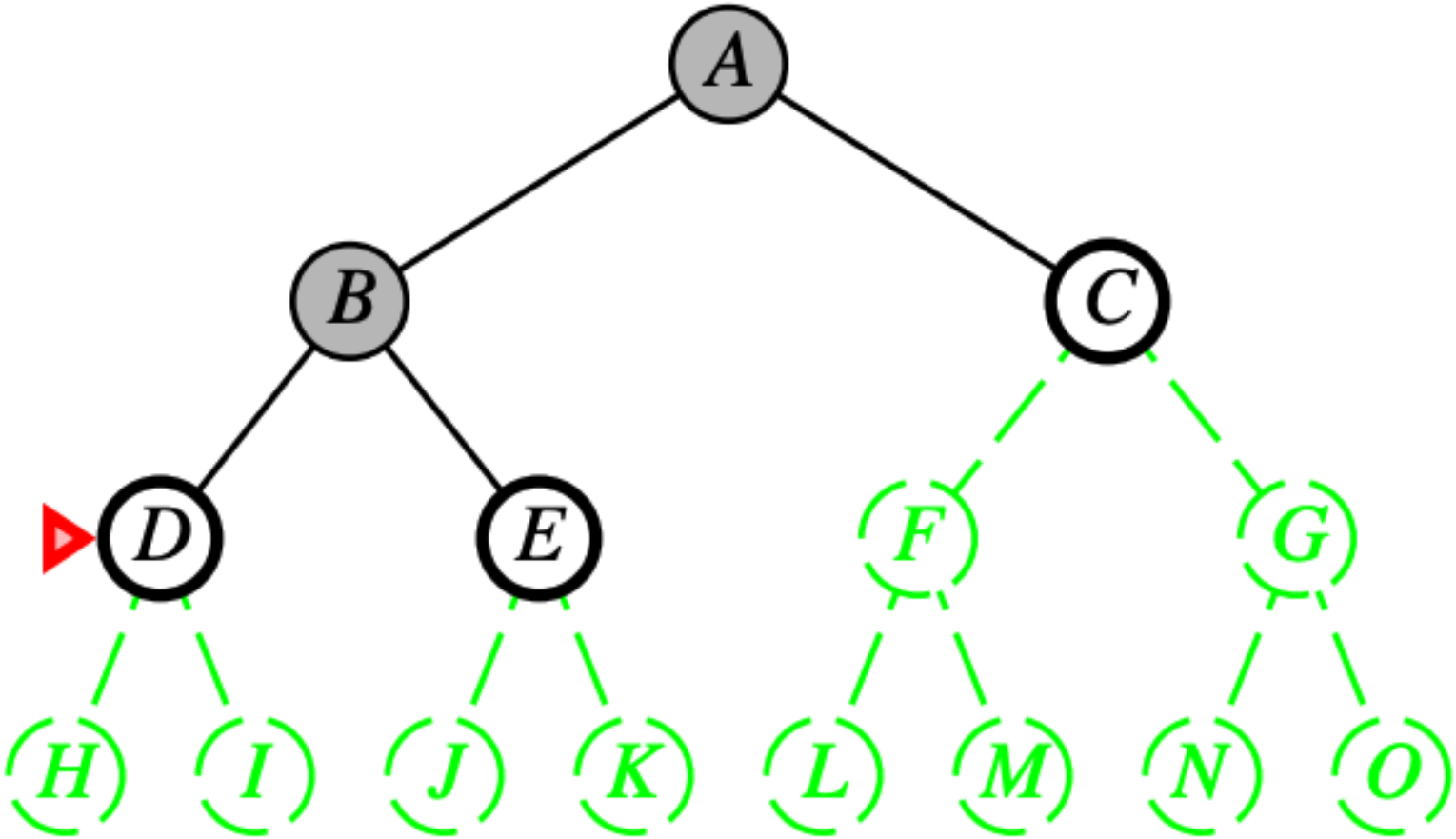
深度优先搜索（depth-first search, DFS）

总是扩展当前边缘节点集中最深的节点



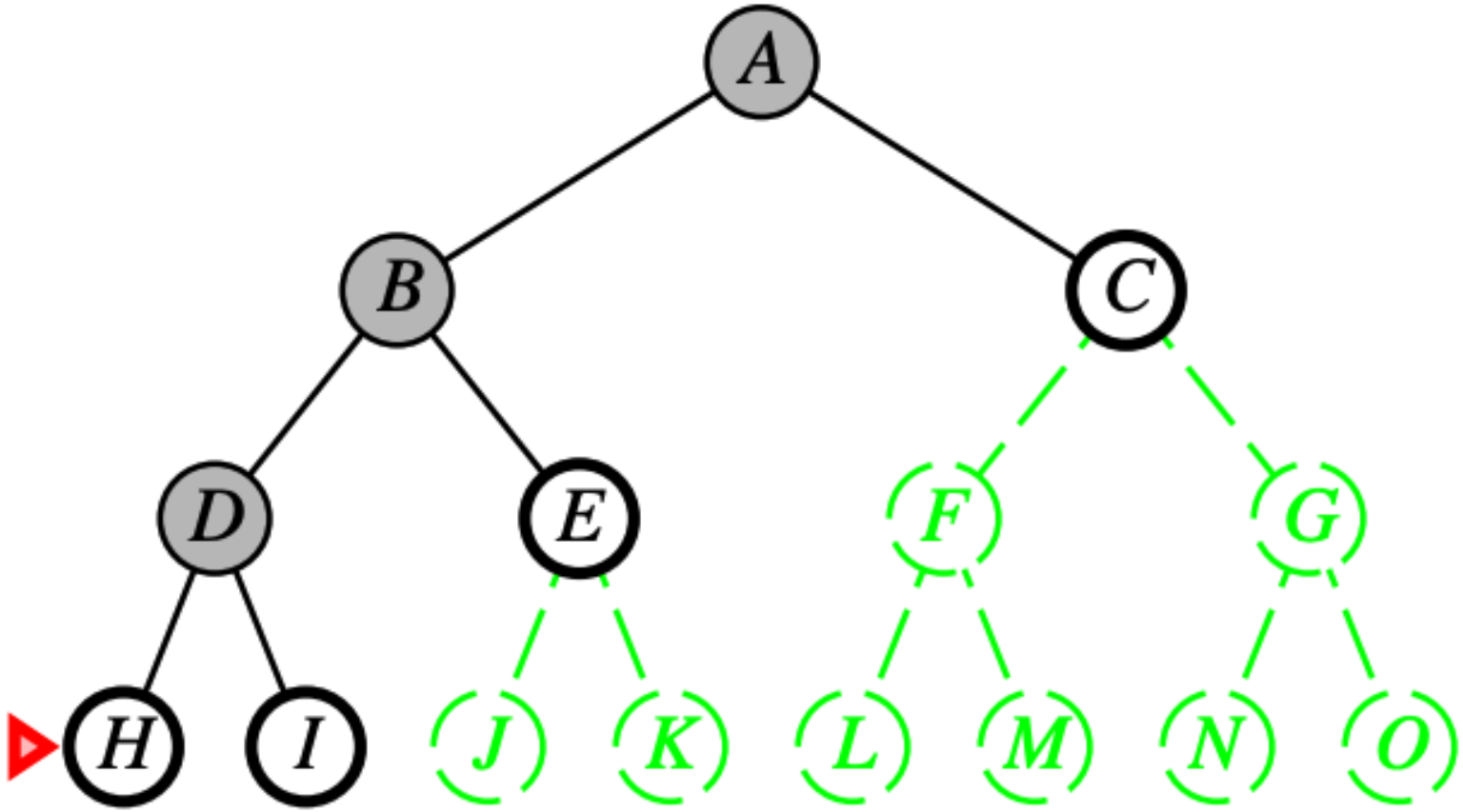
深度优先搜索（depth-first search, DFS）

总是扩展当前边缘节点集中最深的节点



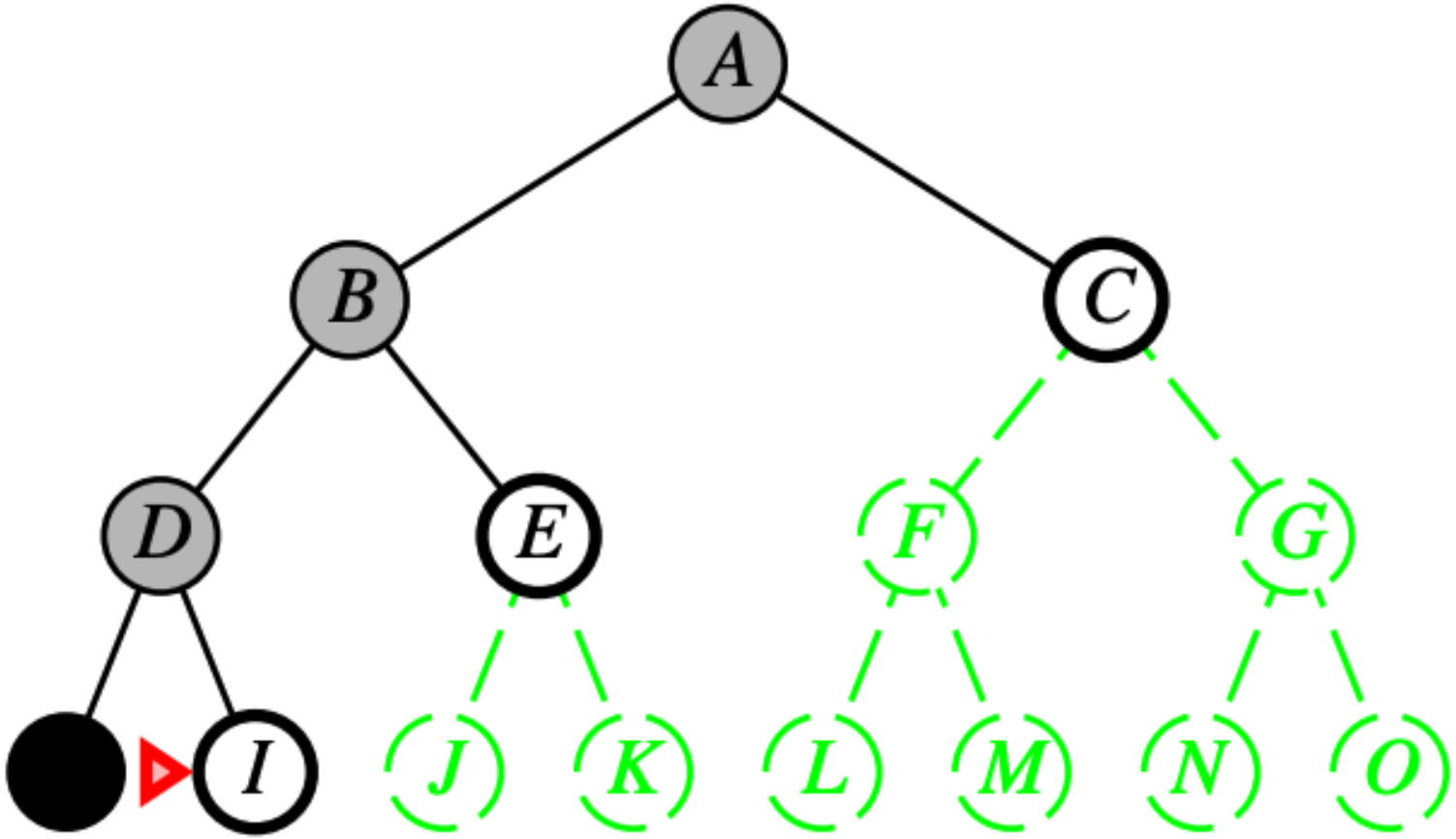
深度优先搜索（depth-first search, DFS）

总是扩展当前边缘节点集中最深的节点



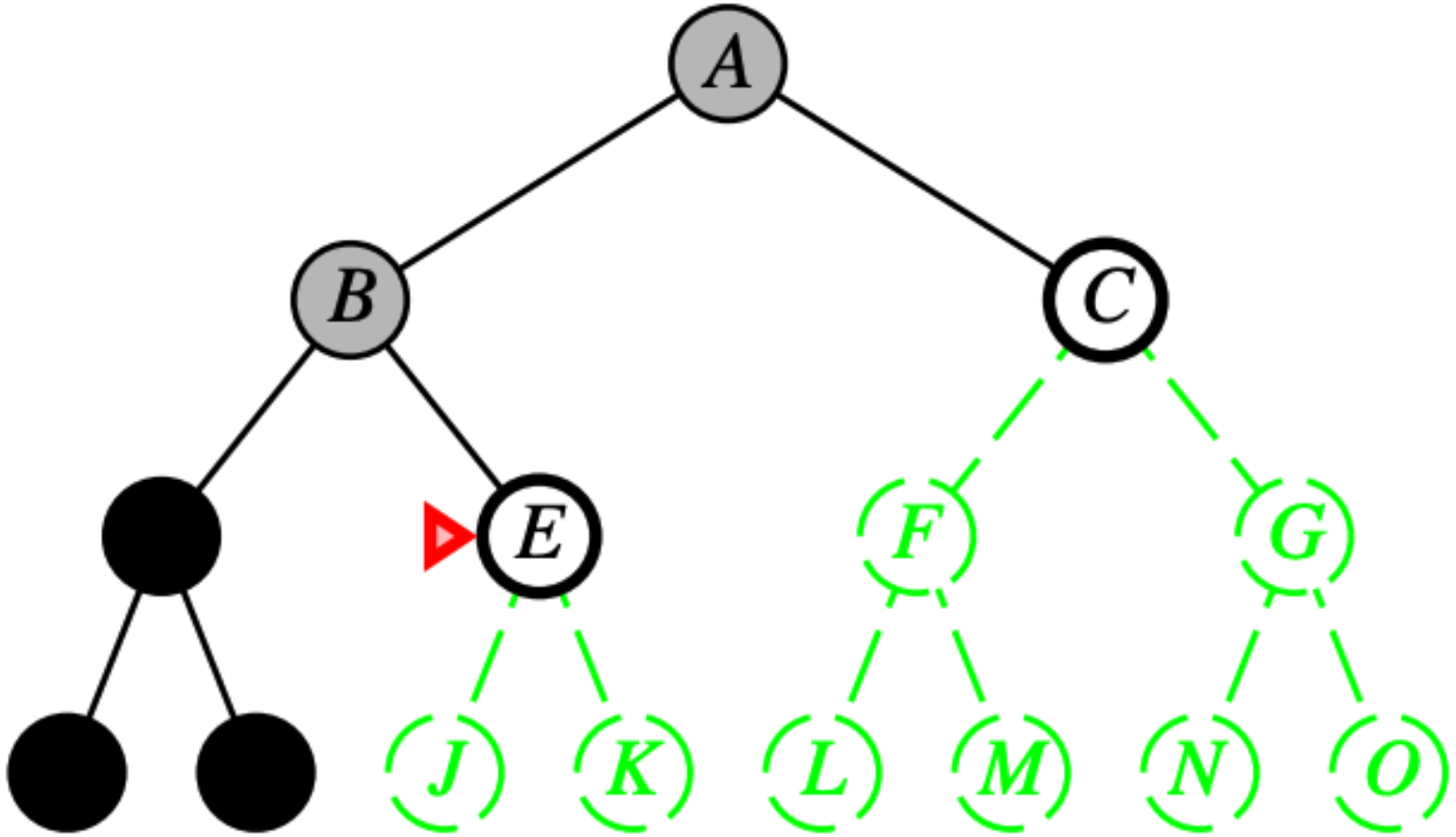
深度优先搜索（depth-first search, DFS）

总是扩展当前边缘节点集中最深的节点



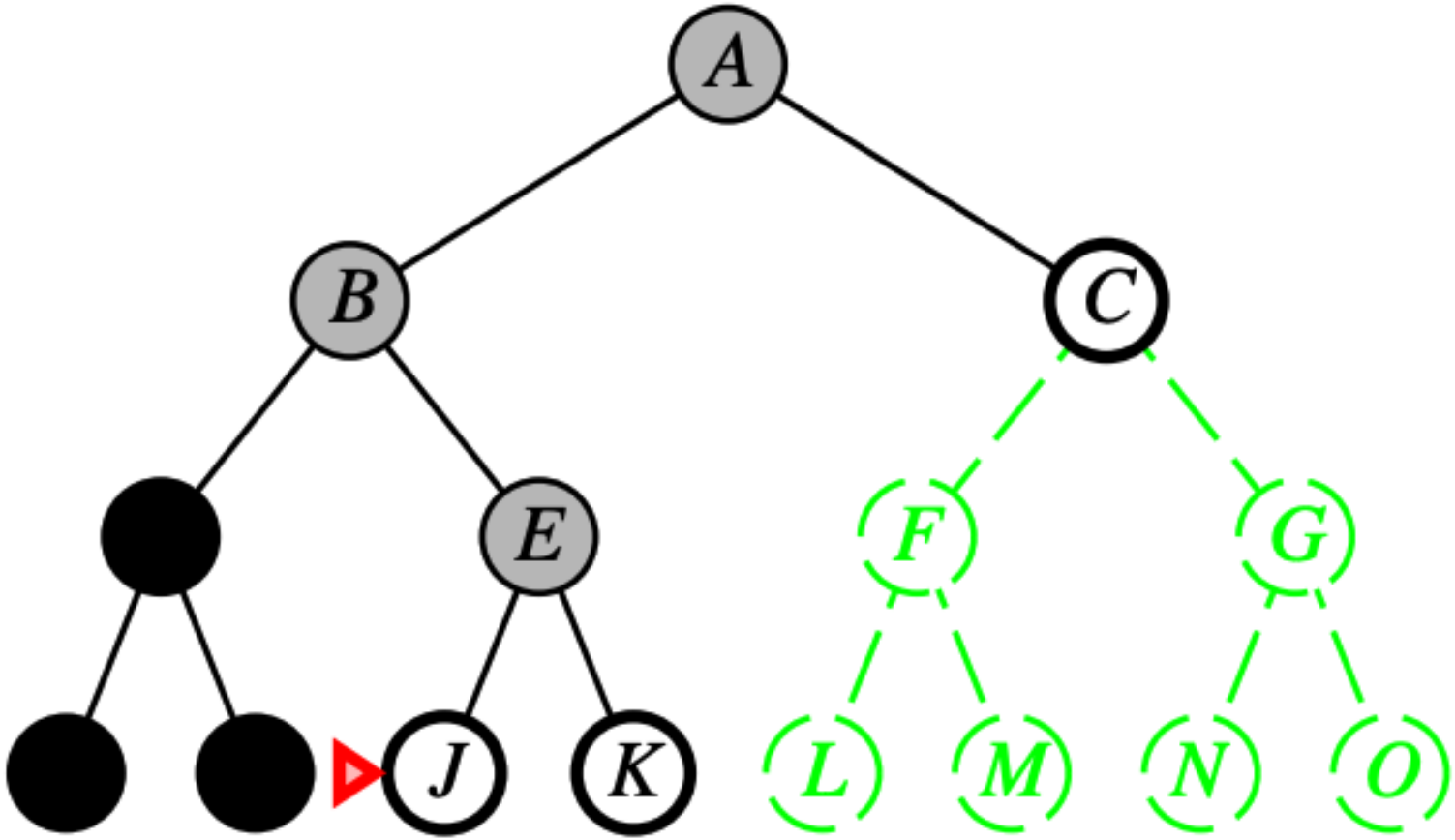
深度优先搜索（depth-first search, DFS）

总是扩展当前边缘节点集中最深的节点



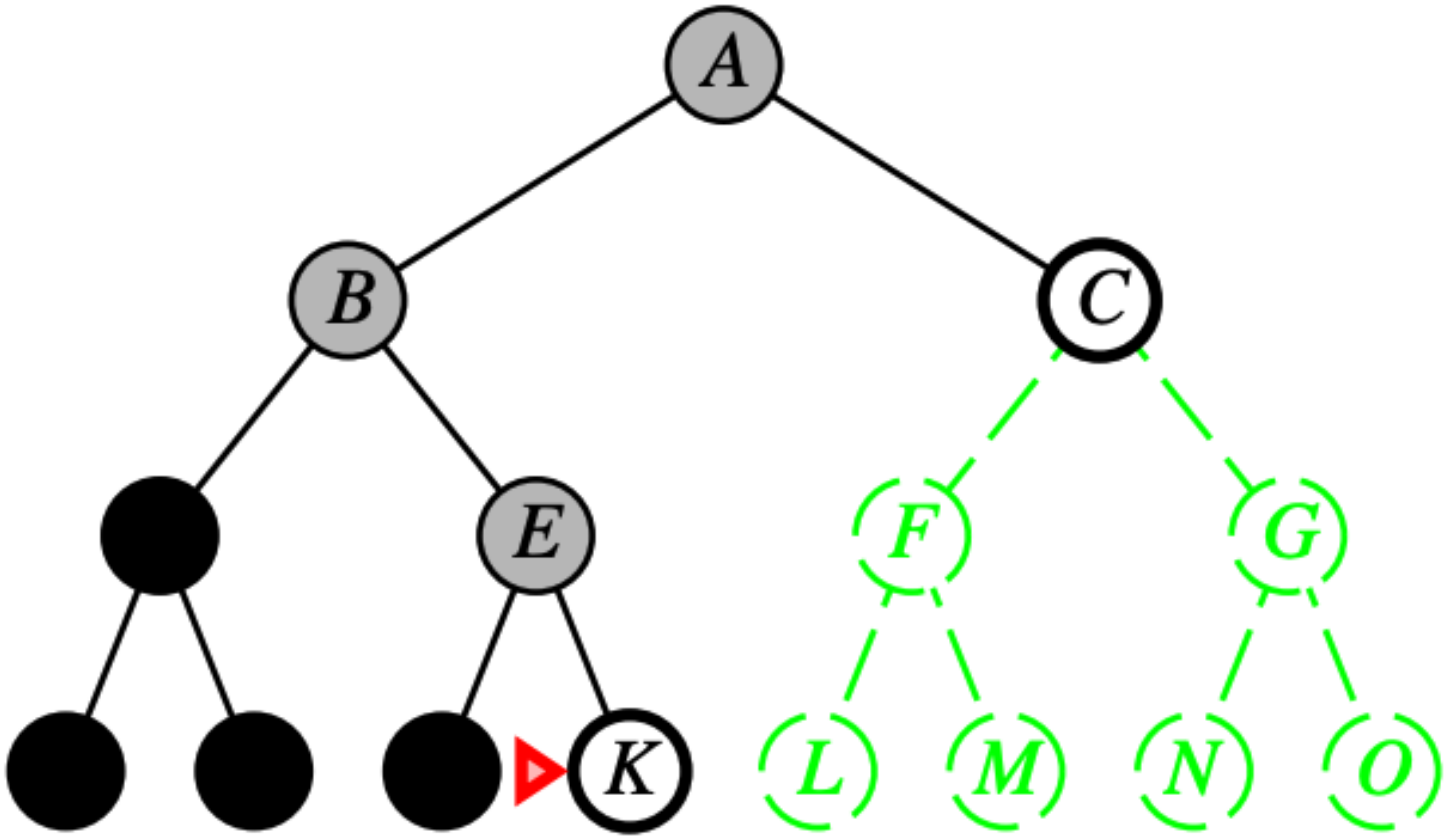
深度优先搜索（depth-first search, DFS）

总是扩展当前边缘节点集中最深的节点



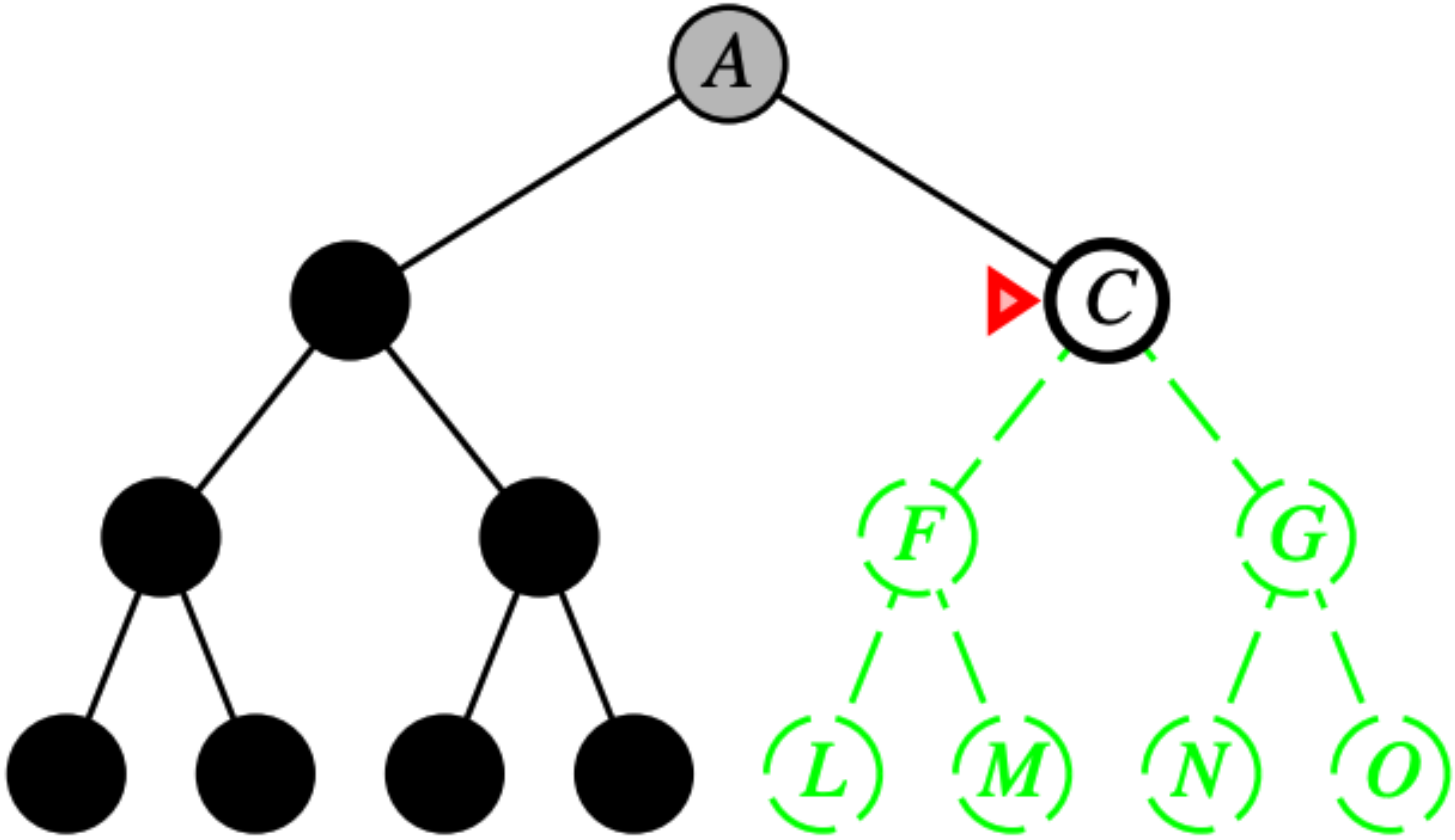
深度优先搜索（depth-first search, DFS）

总是扩展当前边缘节点集中最深的节点

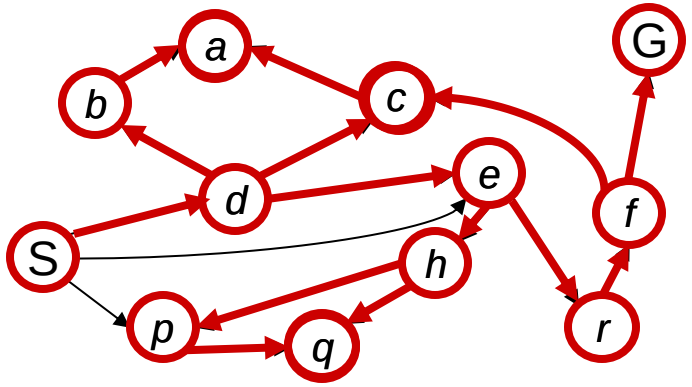


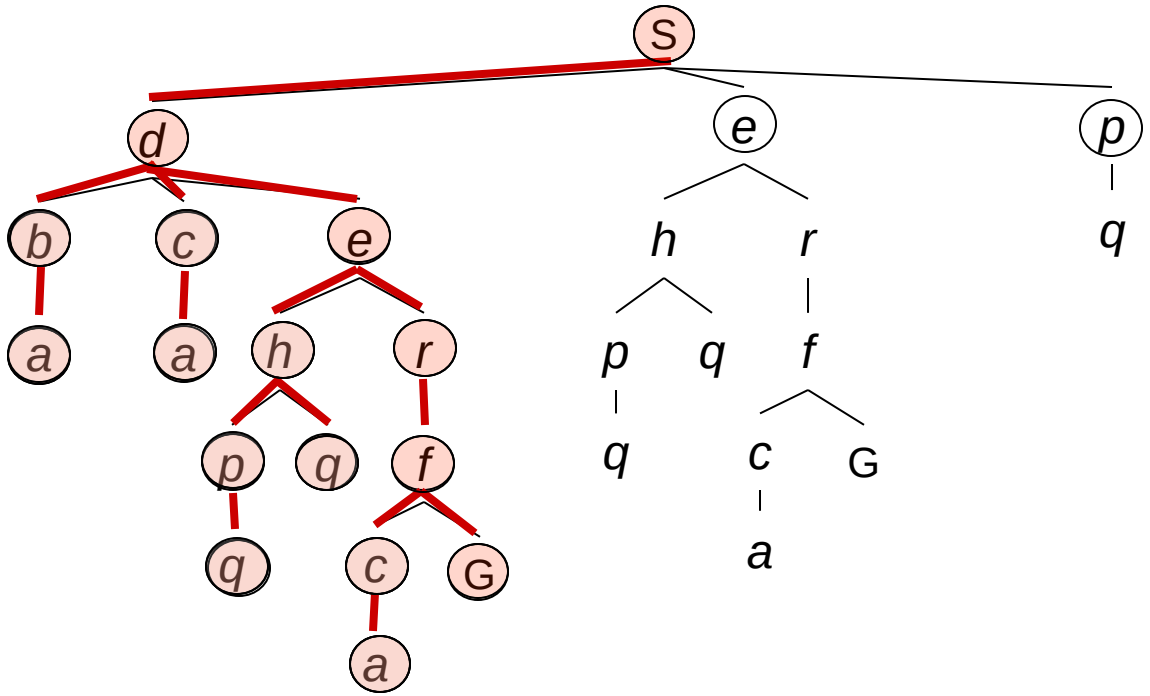
深度优先搜索（depth-first search, DFS）

总是扩展当前边缘节点集中最深的节点

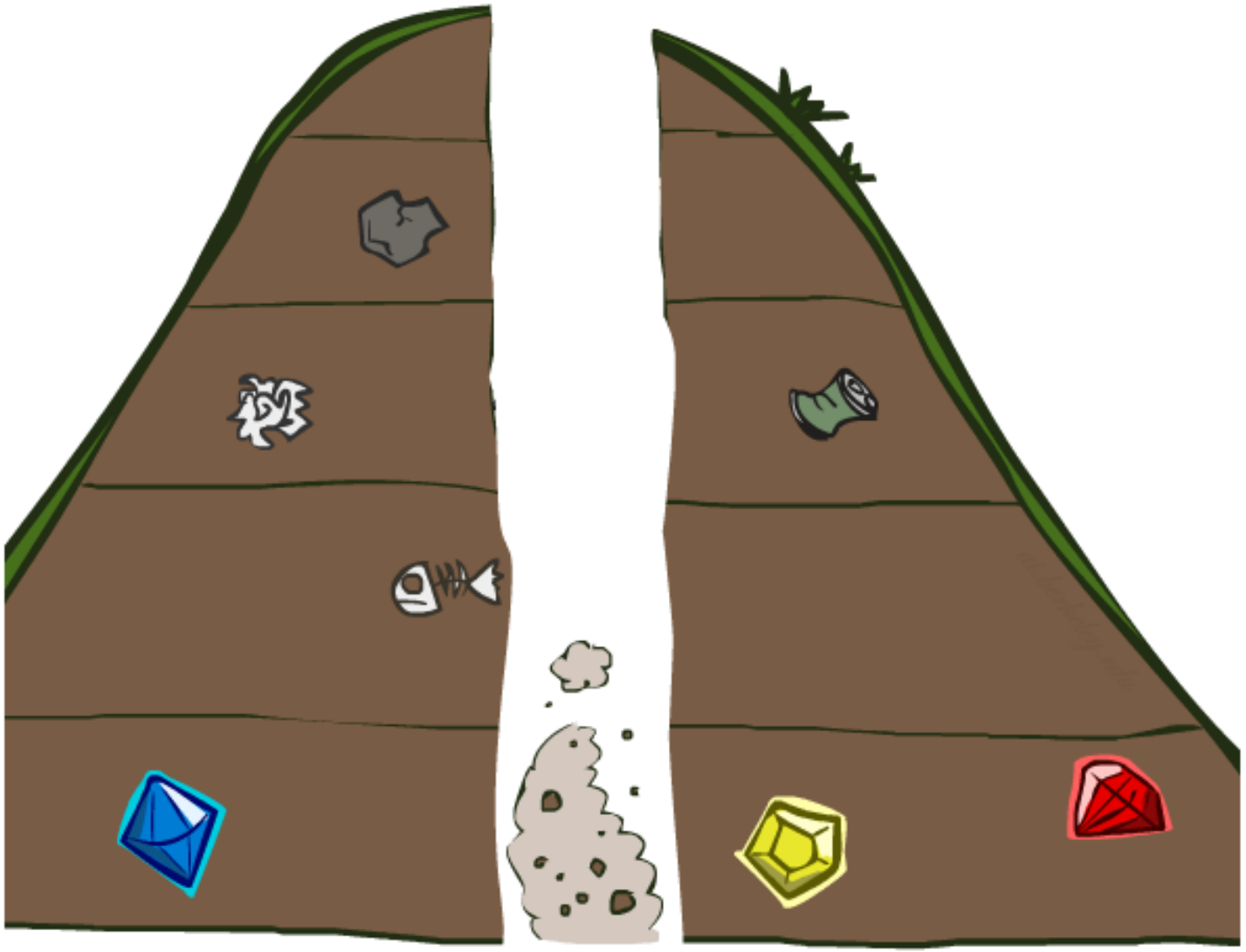


深度优先搜索（depth-first search, DFS）



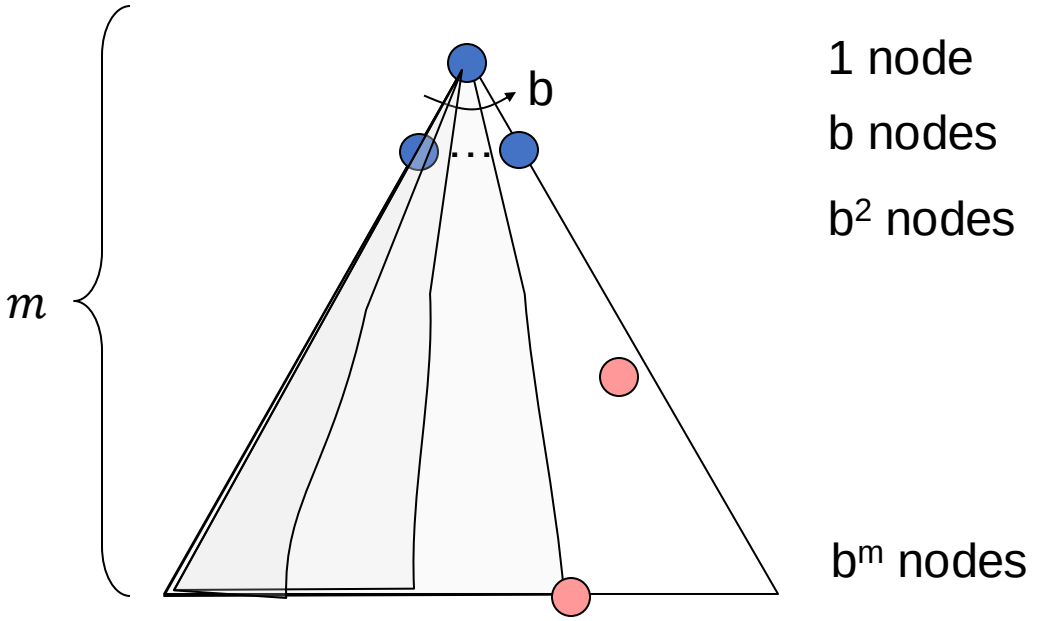


深度优先搜索（depth-first search, DFS）

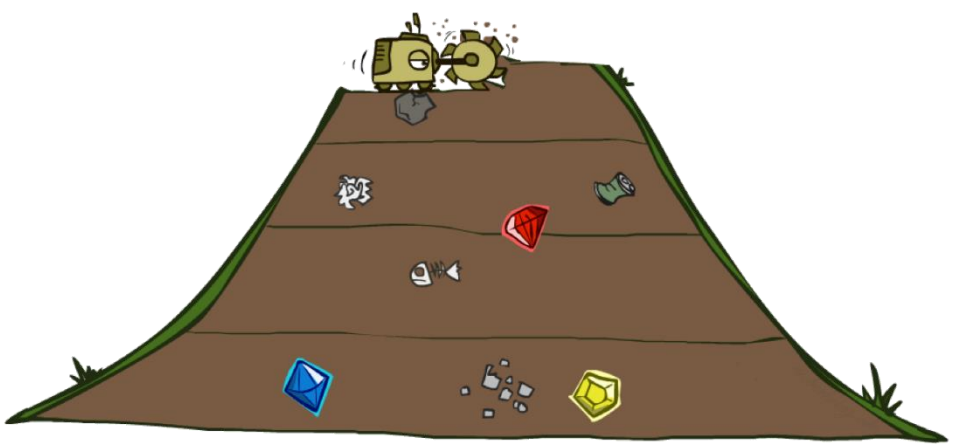


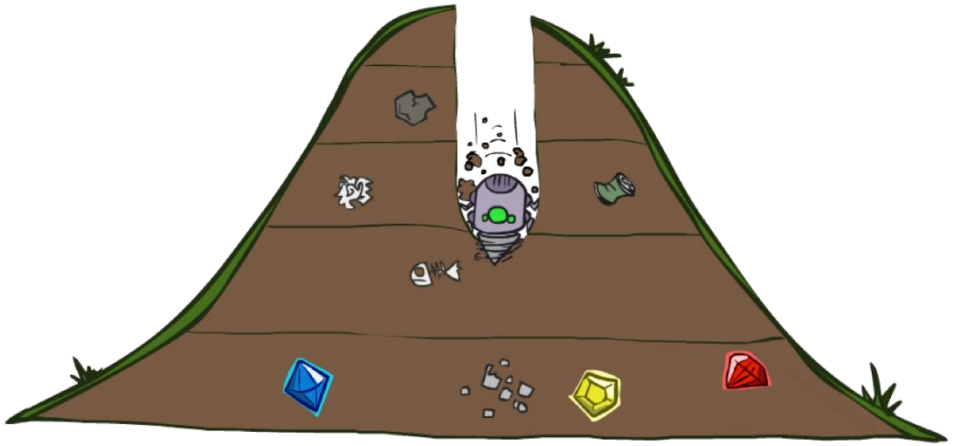
性能分析

- 时间复杂度?
  - $O(b^m)$
- 空间复杂度?
  - 只需要存储一条从根节点到叶节点的路径，以及该路径上每个节点的所有未被扩展的兄弟节点$O(bm)$
- 完备性?
  - No
- 最优性?
  - NO



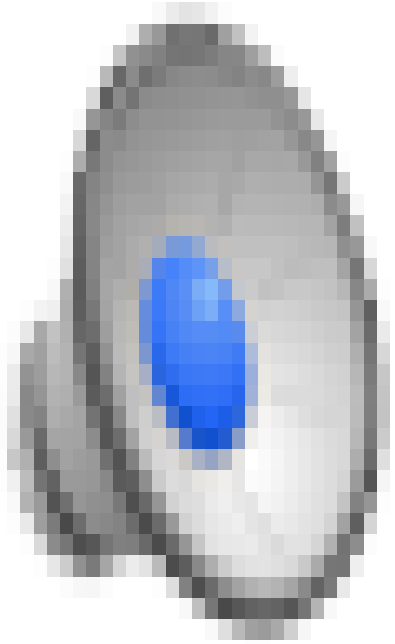
BFS v.s. DFS



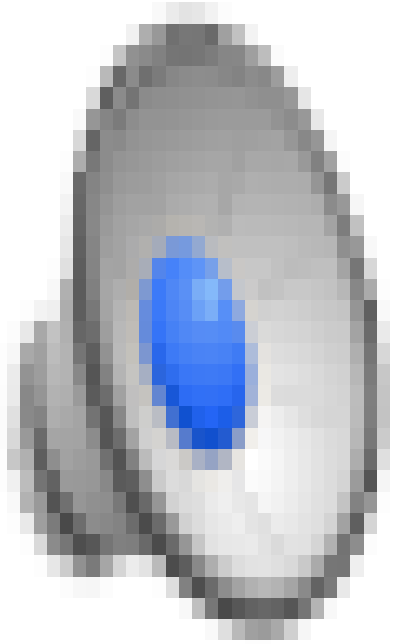


- 什么情况下，DFS比BFS更好
- 什么情况下，BFS比DFS更好

Video of Demo Maze Water DFS/BFS



Video of Demo Maze Water DFS/BFS

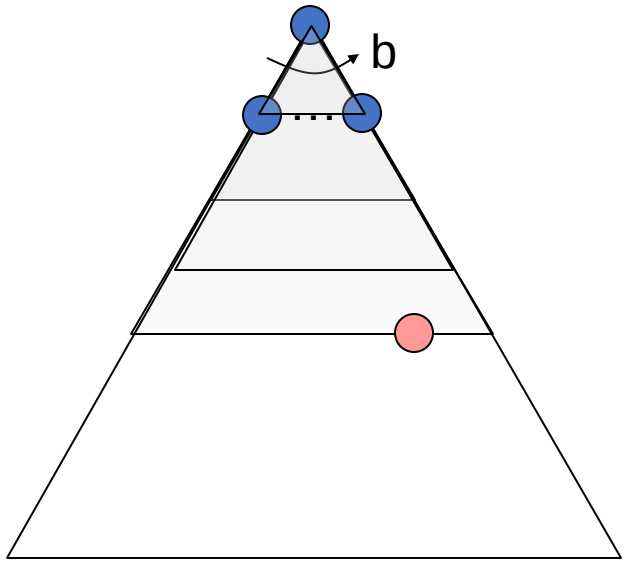


提纲

- 搜索问题
- 无信息搜索
  - 宽度优先搜索
  - 深度优先搜索
  - 迭代加深的深度优先搜索
  - 代价一致性搜索
- 本章小结

迭代加深的深度优先搜索 (iterative deepening search)

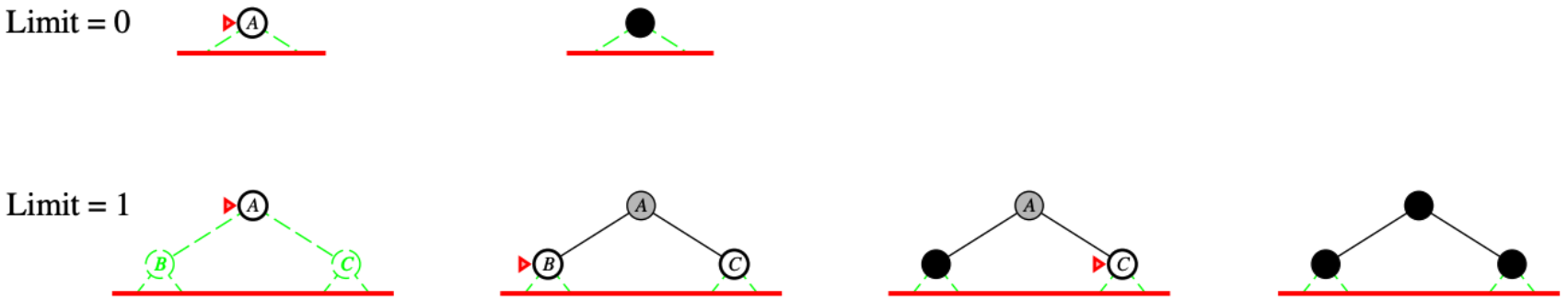
- 能否兼顾DFS和BFS的优势？
  - Run a DFS with depth limit 1. If no solution…
  - Run a DFS with depth limit 2. If no solution…
  - Run a DFS with depth limit 3. …..



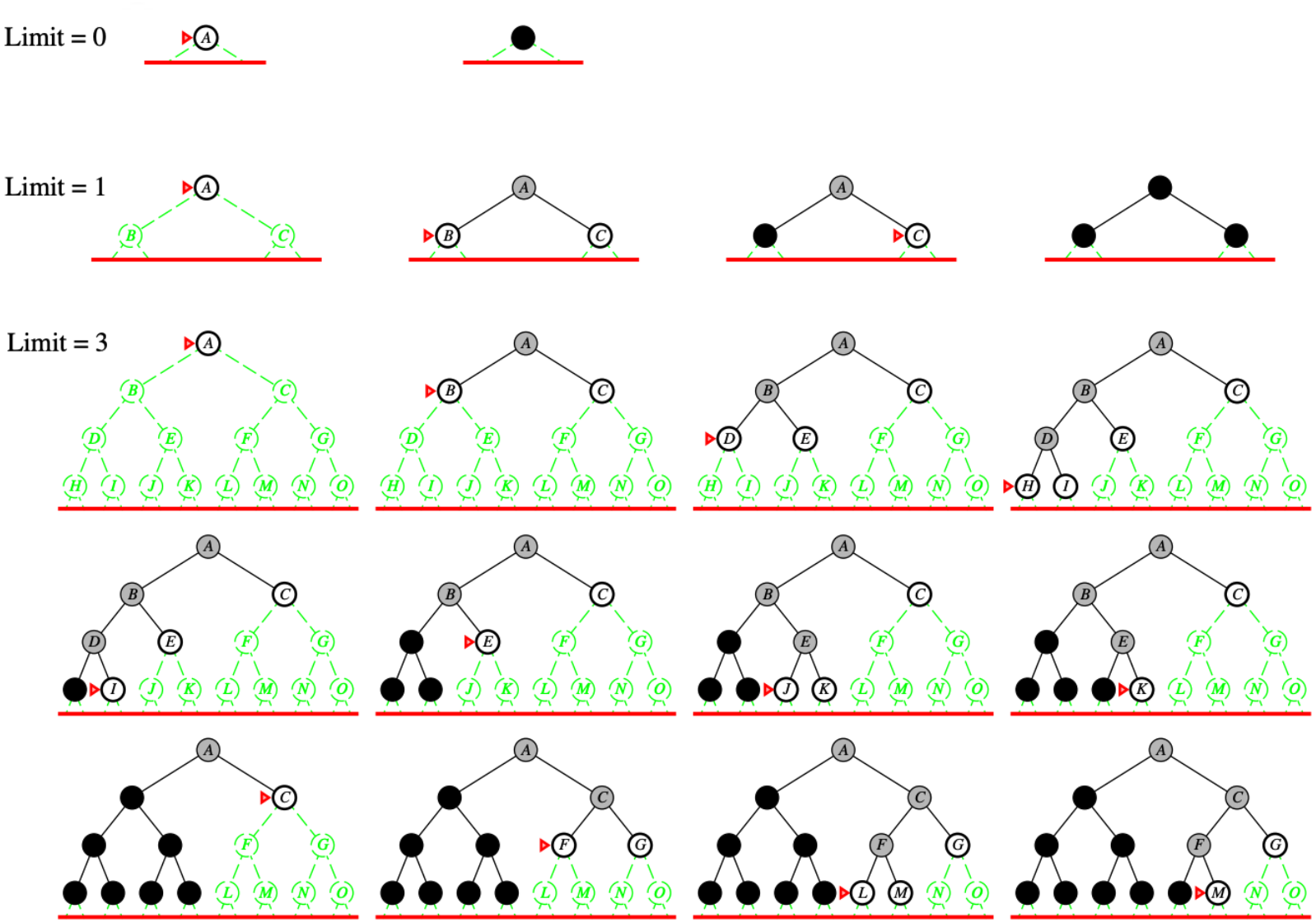
迭代加深的深度优先搜索 (iterative deepening search)

Limit = 0

迭代加深的深度优先搜索 (iterative deepening search)



迭代加深的深度优先搜索 (iterative deepening search)



迭代加深的深度优先搜索 (iterative deepening search)

部分节点被多次生成，是否会造成浪费？

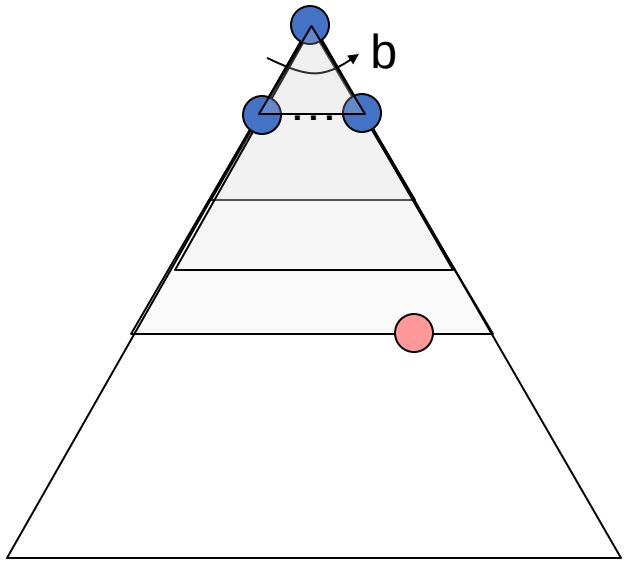
在迭代加深的深度优先搜索中，底层（深度d）节点只被生成一次，倒数第二层的节点被生成两次，以此类推，一直到根节点，被生成d次

$$d(b) + (d-1)b^2 + \cdots (1)b^d$$

绝大多数重复的节点都在上层，上层节点的重复生成影响不大

性能分析

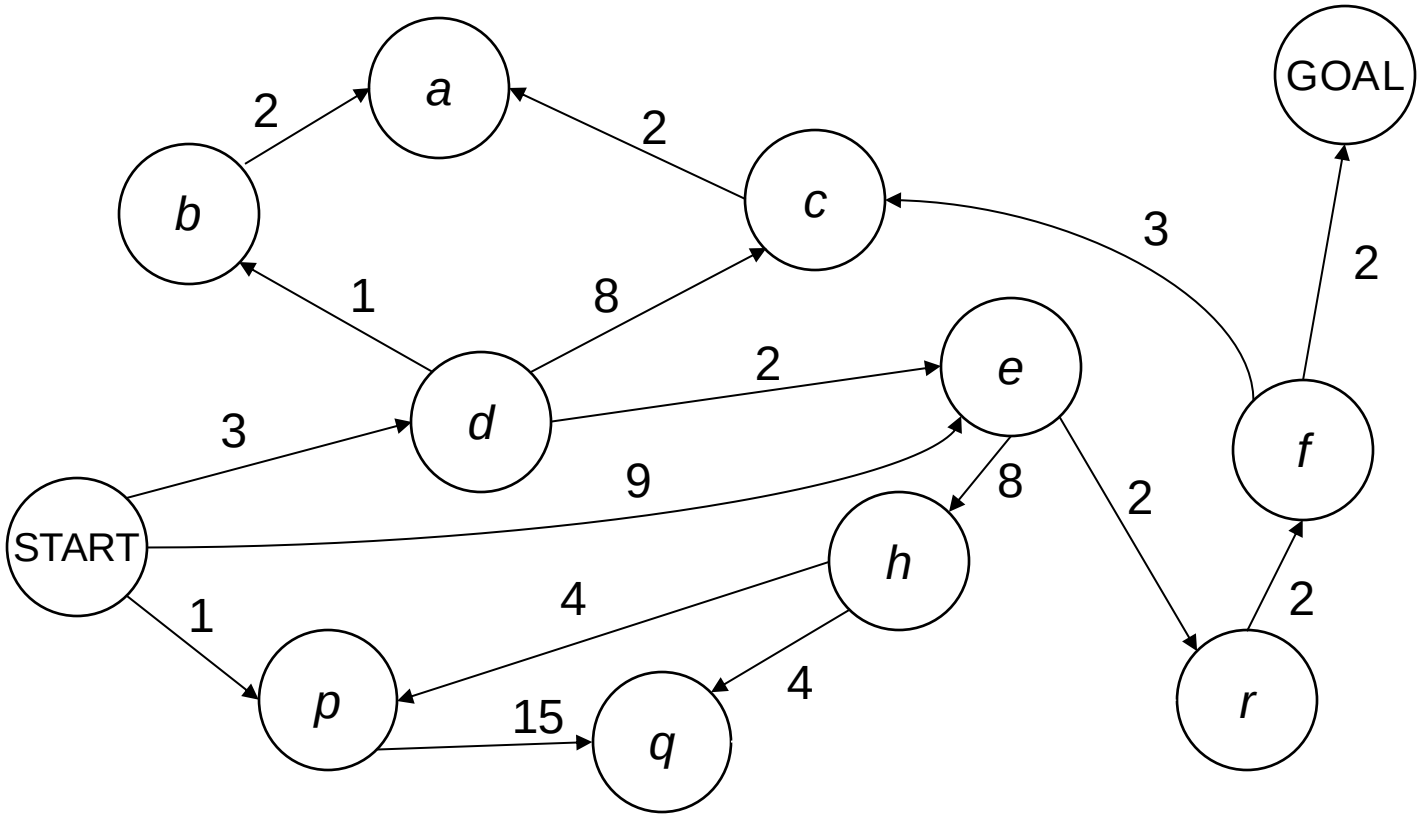
- 时间复杂度?
  - $O(b^d)$
- 空间复杂度?
  - $O(bd)$
- 完备性?
  - Yes
- 最优性?
  - Yes，如果所有行动代价相同



提纲

- 搜索问题
- 无信息搜索
  - 宽度优先搜索
  - 深度优先搜索
  - 迭代加深的深度优先搜索
  - 一致性代价搜索
- 本章小结

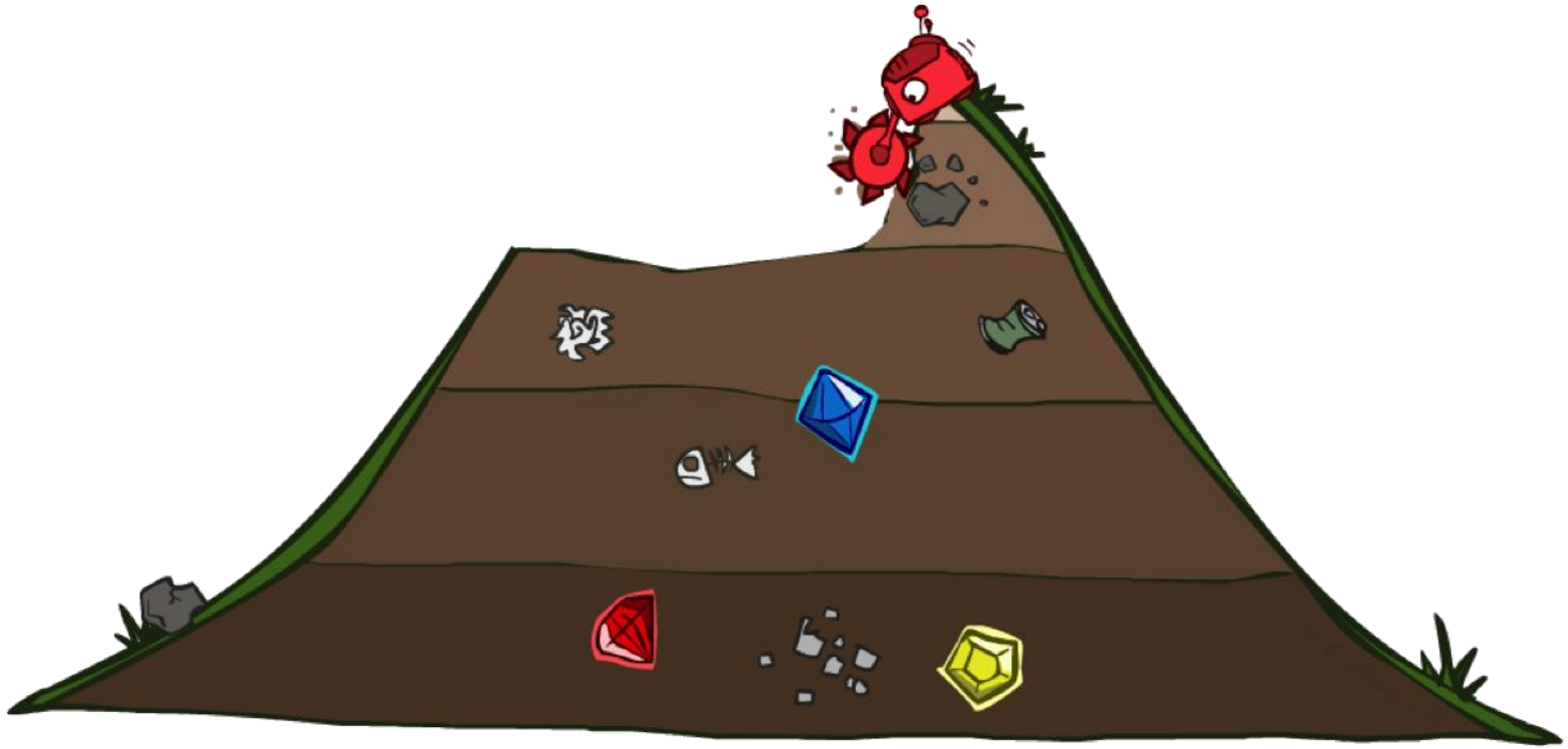
代价敏感搜索



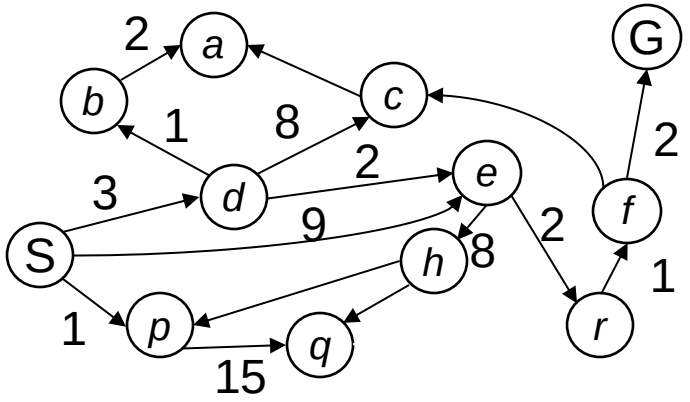
如何寻找代价最小的路径？

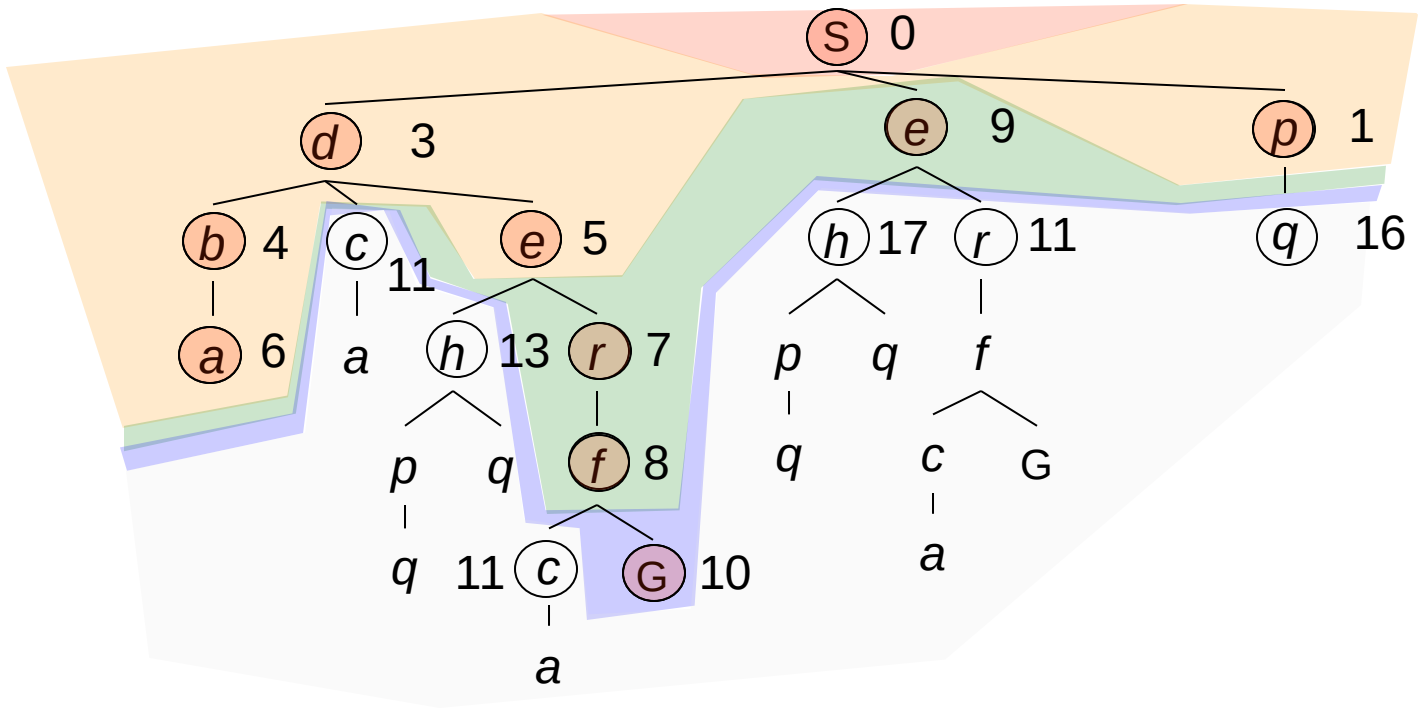
一致代价搜索(uniform-cost search)

一致代价搜索扩展路径消耗最小的节点



一致代价搜索(uniform-cost search)

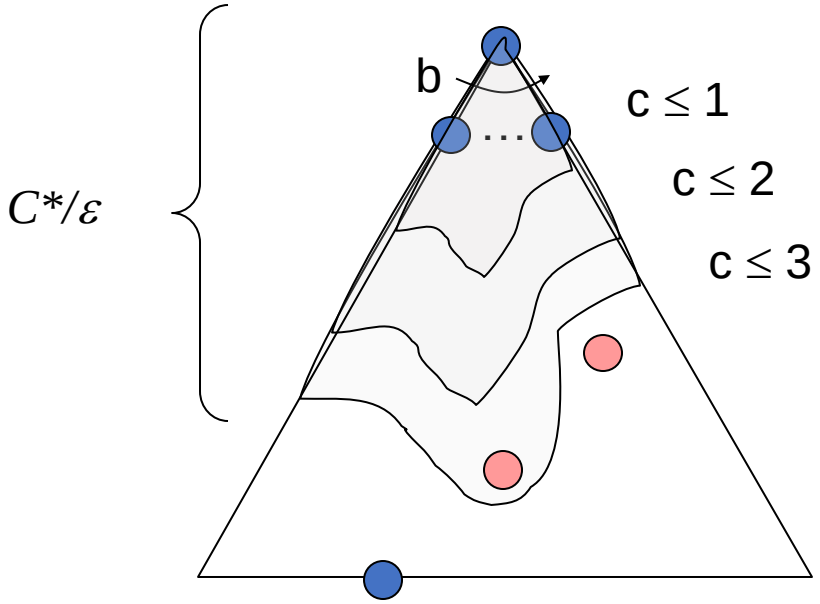




性能分析

假设最优解代价为C^*， ϵ为每个动作代价的下界

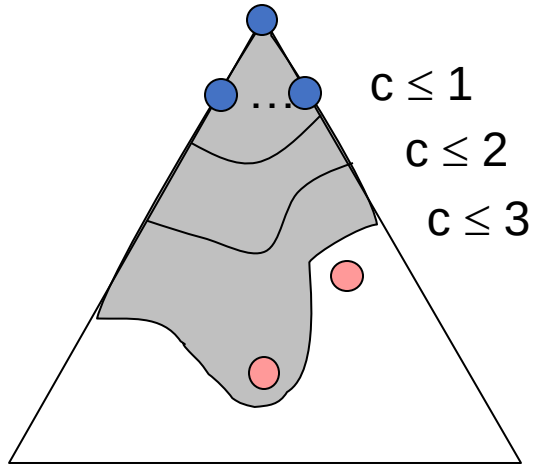
- 时间复杂度?
  - $O(b^{\frac{C^*}{\epsilon}})$
- 空间复杂度?
  - $O(b^{\frac{C^*}{\epsilon}})$
- 完备性?
  - Yes，如果每一步的代价≥ $\epsilon(\epsilon > 0)$
- 最优性?
  - Yes

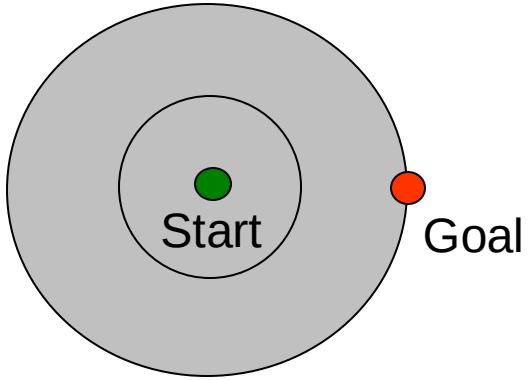


存在的问题

- UCS 按照路径代价进行探索

- 局限：
  - 需要在所有方向上进行探索
  - 没有利用关于目标状态的信息





提纲

- 搜索问题
- 无信息搜索
  - 宽度优先搜索
  - 深度优先搜索
  - 迭代加深的深度优先搜索
  - 一致性代价搜索
- 本章小结

本章小结

- 搜索问题：初始状态、动作、状态转移、目标测试、路径代价
- 宽度优先搜索：逐层扩展节点
- 深度优先搜索：优先扩展最深的节点
- 迭代加深的深度优先搜索：限制深度的DFS
- 一致性代价搜索：优先扩展路径代价最小的节点