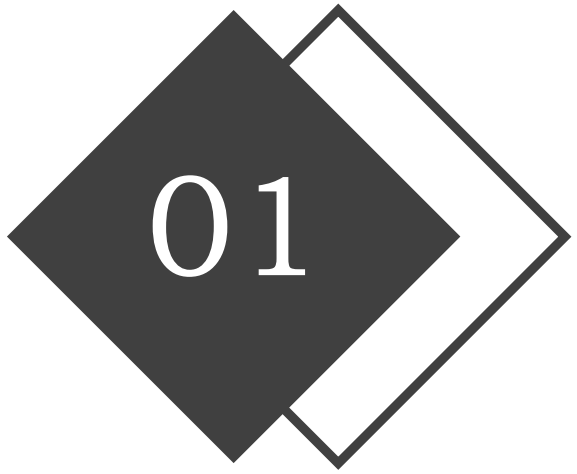
无信息搜索

陈姝睿　2021.10.09

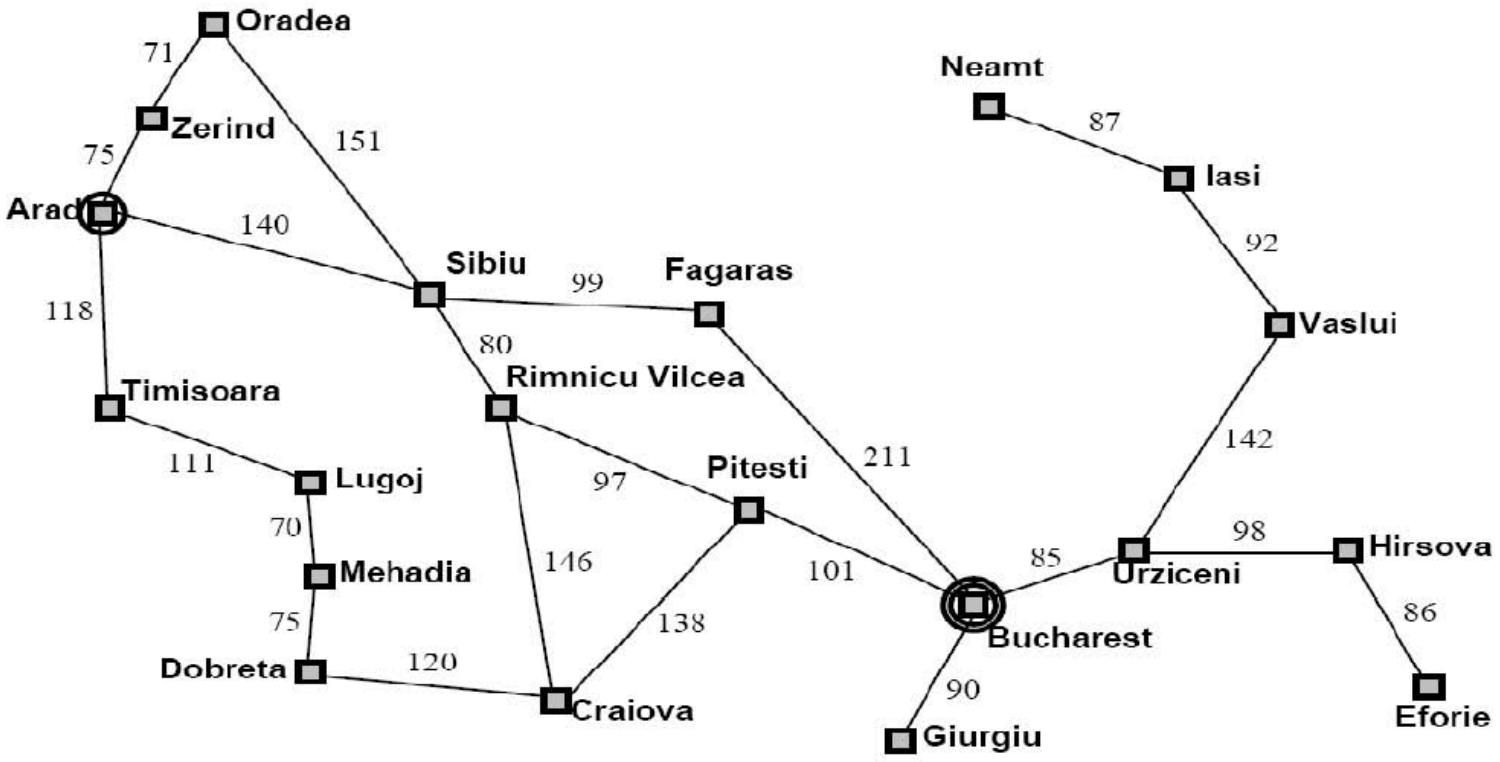
目录



搜索问题定义

举例

Currently in Arad, need to get to Bucharest

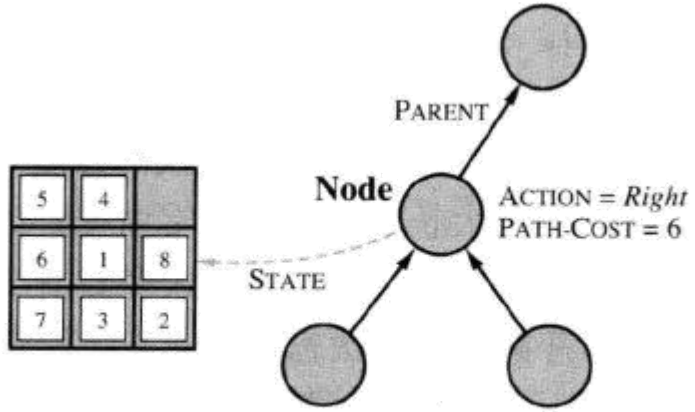


- **States:** the various cities you could be located in.
- **Actions:** drive between neighboring cities.
- **Initial state:** in Arad
- **Goal:** in Bucharest
- **Solution:** the route, the sequence of cities to travel through to get to Bucharest.

搜索的数据结构

对树中每个结点n，一般定义如下数据结构

- *n*.STATE：对应状态空间中的状态；
- *n*.PARENT：搜索树中产生该结点的结点（即父结点）；
- *n*.ACTION：父结点生成该结点时所采取的行动；
- *n*.PATH-COST：代价，一般用 $g(n)$表示，指从初始状态到达该结点的路径消耗；

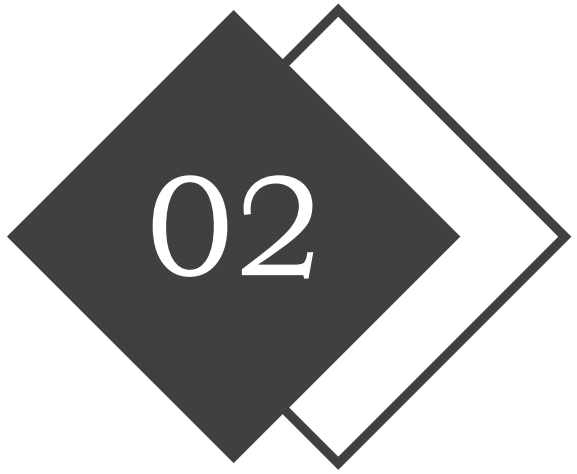


结点是数据结构，由搜索树构造。每个结点都有一个父结点、一个状态和其他域。箭头由子结点指向父结点

求解算法的性能

四个方面：

- 完备性：当问题有解时，这个算法能否保证找到解。
- 最优性：搜索策略能否找到最优解。
- 时间复杂度：找到解所需要的时间，也叫搜索代价
- 空间复杂度：执行搜索过程中需要多少内存空间

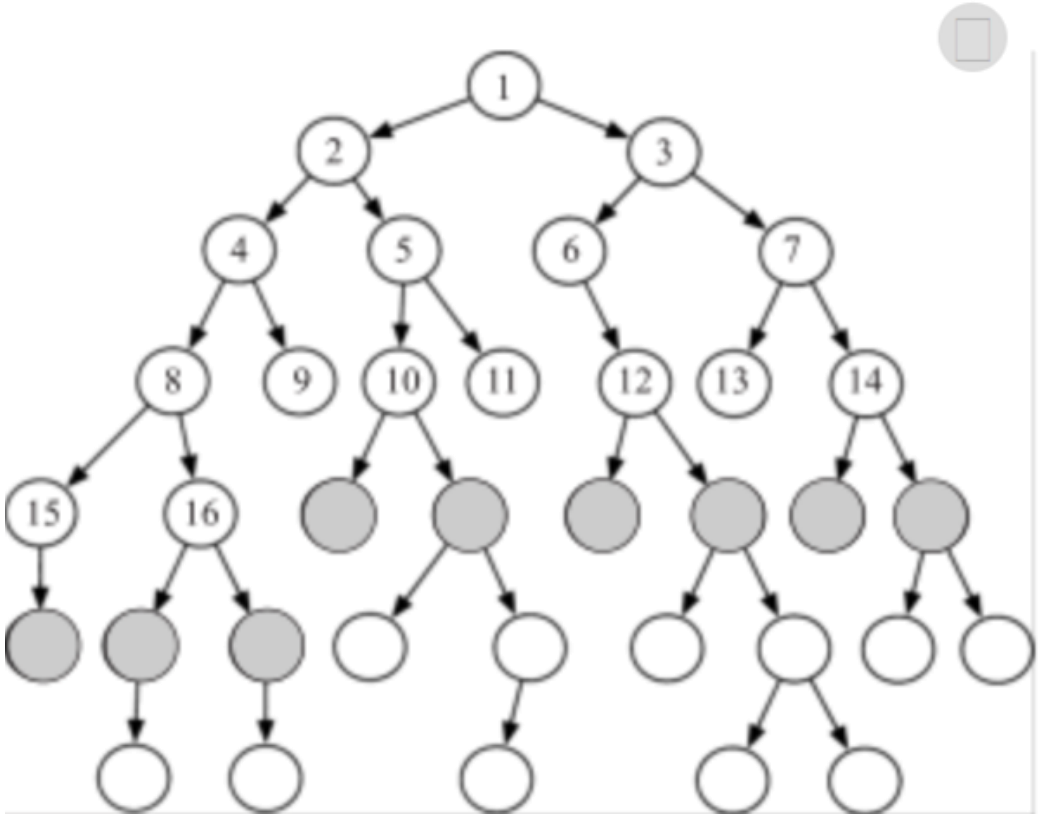


无信息搜索策略

无信息搜索策略

- 宽度优先搜索
- 一致代价搜索
- 深度优先搜索
- 深度受限搜索
- 迭代加深搜索
- 双向搜索

宽度优先搜索(BFS)



- 完备性;
- 非最优;
- 时间复杂度O(b^d);
- 空间复杂度O(b^d);

- 节点扩展顺序与目标节点的位置无关;
- 用一个先进先出（FIFO）队列实现;

一致代价搜索Uniform-cost search (UCS)

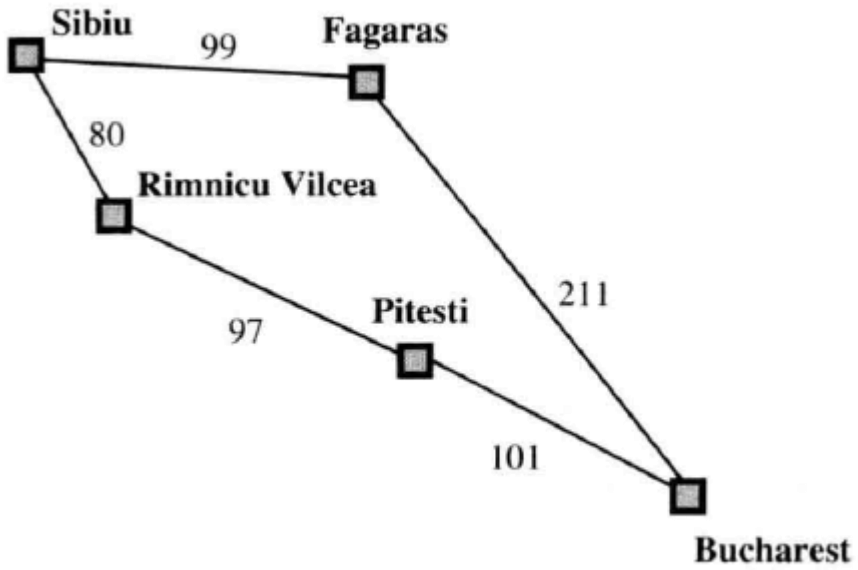


图 3.15　罗马尼亚问题的部分状态空间，用于描述一致代价搜索

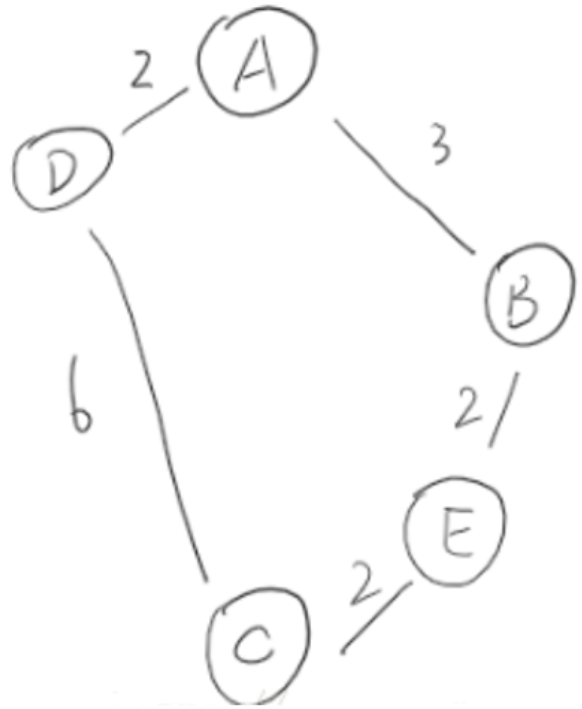
Search Strategy： 扩展最低代价的未扩展节点。

Implementation： 队列，按路径代价排序，最低优先。

Complexity： $O(b^{1+\lfloor C^*/e \rfloor})$

无信息搜索策略

```
function UNIFORM-COST-SEARCH(problem) returns a solution, or failure
  node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
  frontier ← a priority queue ordered by PATH-COST, with node as the only element
  explored ← an empty set
  loop do
    if EMPTY?(frontier) then return failure
    node ← POP(frontier)  /* chooses the lowest-cost node in frontier */
    if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
    add node.STATE to explored
    for each action in problem.ACTIONS(node.STATE) do
      child ← CHILD-NODE(problem, node, action)
      if child.STATE is not in explored or frontier then
        frontier ← INSERT(child, frontier)
      else if child.STATE is in frontier with higher PATH-COST then
        replace that frontier node with child
```



深度优先搜索 (DFS)

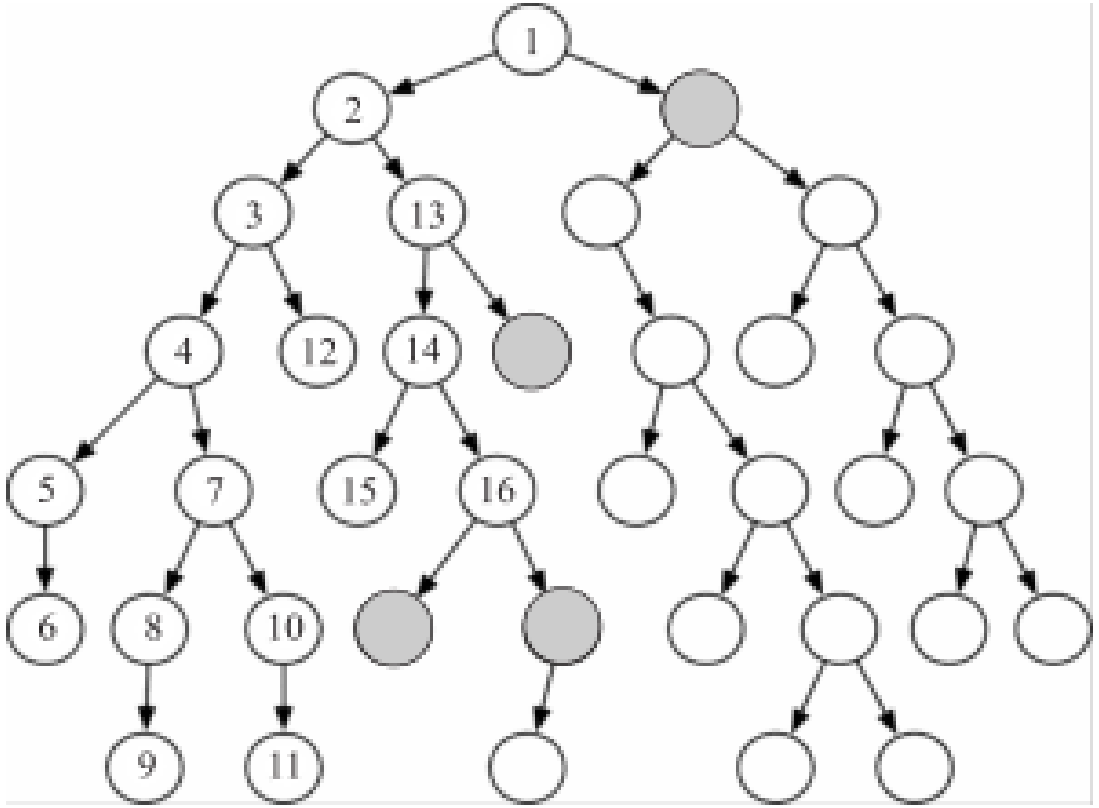


图 3-5　深度优先搜索中节点的扩展顺序

深度受限搜索 (Depth-limited Search)

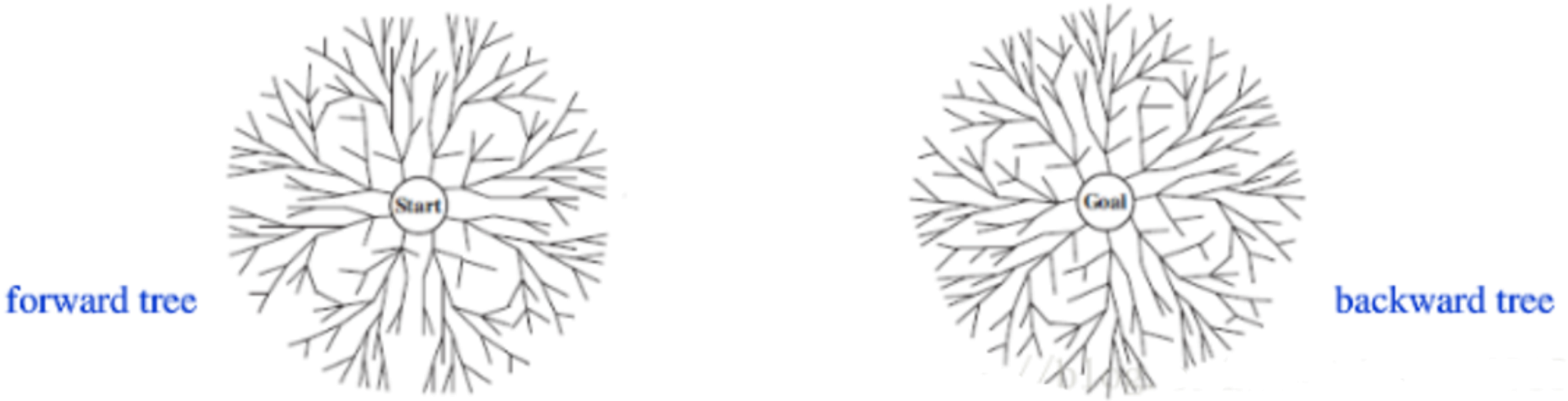
```
function Depth-Limited-Search(problem, limit) returns a solution, or failure/cutoff
    if problem.Goal-Test(node.State) then return Solution(node)
    if limit = 0 then return cutoff     //no solution
    cutoff_occurred? ← false
    for each action in problem.Action(node.State) do
        child ← Child-Node(problem, node, action)
        result ← Recusive-DLS(child, problem, limit - 1)
        if result = cutoff then cutoff_occurred ? ← true
        else if result ≠ failure then return result
    if cutoff_occurred ? then return cutoff    //no solution
    else return failure
```

迭代加深搜索（Iterative Deepening Search ）

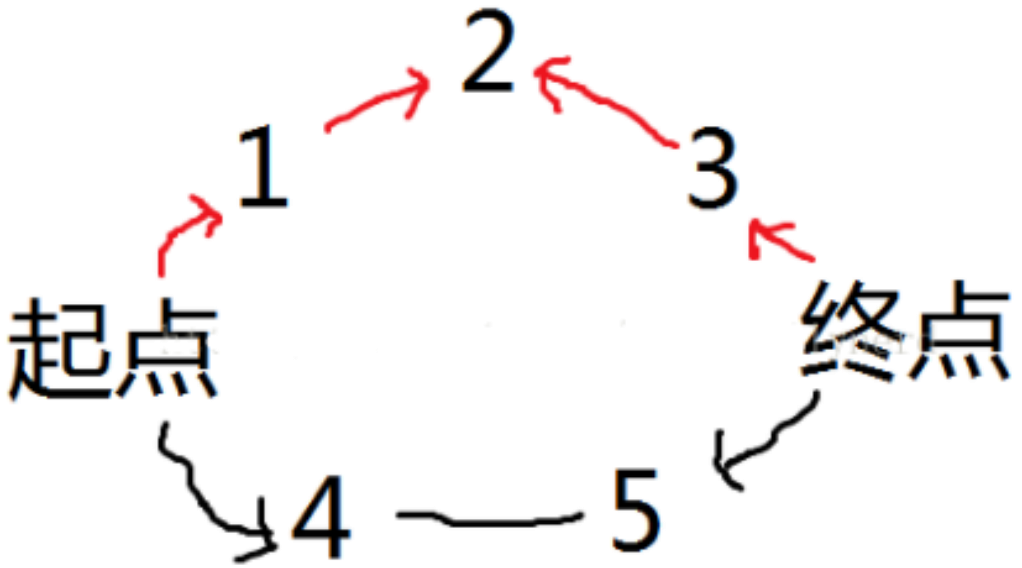
```
1  function Iterative-Deepening-Search(problem, limit) returns a solution, or failure
2      for depth = 0 to ∞ do
3          result ← Depth-Limited-Search(problem,  depth)
4          if result ≠ cutoff then return result
```

双向搜索 （Bidirectional search）

它同时进行两个搜索：一个是从初始状态向前搜索，二另一个则从目标向后搜索。当两者在中间相遇时停止。



双向搜索 （Bidirectional search）



无信息树搜索策略评价

Evaluation of Uninformed Tree-search Strategies

无信息树搜索策略评价

Criterion	Breadth First	Uniform Cost	Depth First	Depth Limited	Iterative Deepening	Bidirectional
Complete	Yes^{a}	Yes^{a,b}	No	No	Yes^{a}	Yes^{a,d}
Time	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(b^m)$	$O(b^\ell)$	$O(b^d)$	$O(b^{d/2})$
Space	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(bm)$	$O(b\ell)$	$O(bd)$	$O(b^{d/2})$
Optimal	Yes^{c}	Yes	No	No	Yes^{c}	Yes^{c,d}

Where

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

- a -- complete if b is finite
- b -- complete if step costs ϵ for positive
- c -- optimal if step costs are all identical
- d -- if both directions use breadth-first search