无信息搜索&启发式搜索

陈姝睿　2019.11.15

搜索问题定义

搜 索 问 题 定 义

我们需要考虑下面几个部分来对搜索问题进行形式化定义：

- 状态空间(state space)：表示需要进行搜索的空间。状态空间是对问题的形式化
- 动作(action)：表示从一个状态到另一个状态。动作是对真实的动作的形式化
- 初始状态(initial state)：当前状态的表示
- 目标(goal)：需要达到的目标状态的表示
- 启发方法(heuristics)：用于指挥搜索的前进方向

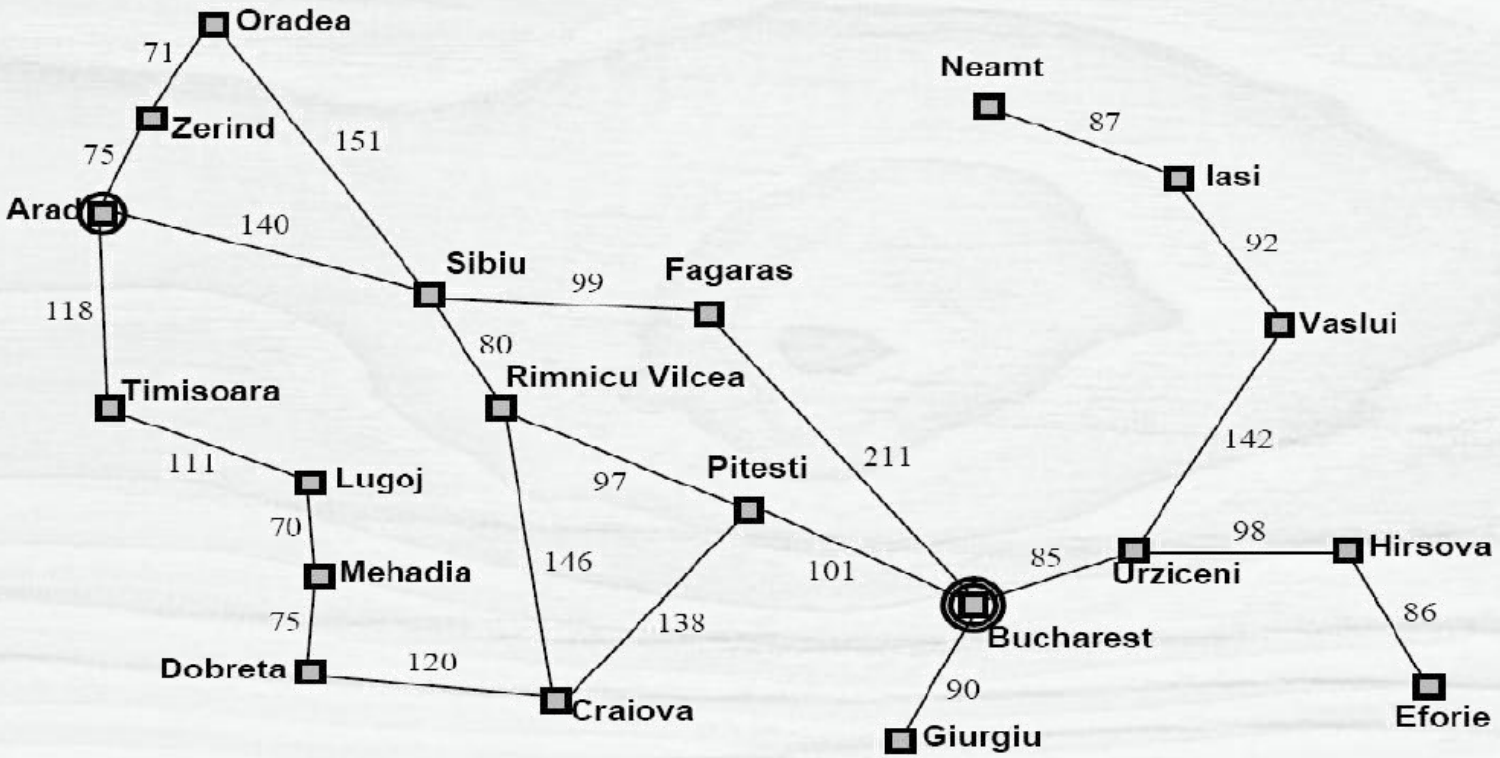
问题的解(solution)的定义

A solution to a problem is an action sequence that leads from the initial state to a goal state.

一个问题的解是一个动作序列。

举例

Currently in Arad, need to get to Bucharest

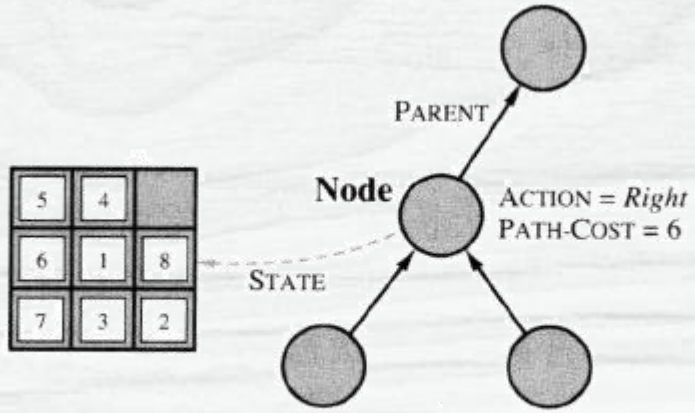


- **States:** the various cities you could be located in.
- **Actions:** drive between neighboring cities.
- **Initial state:** in Arad
- **Goal:** in Bucharest
- **Solution:** the route, the sequence of cities to travel through to get to Bucharest.

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：

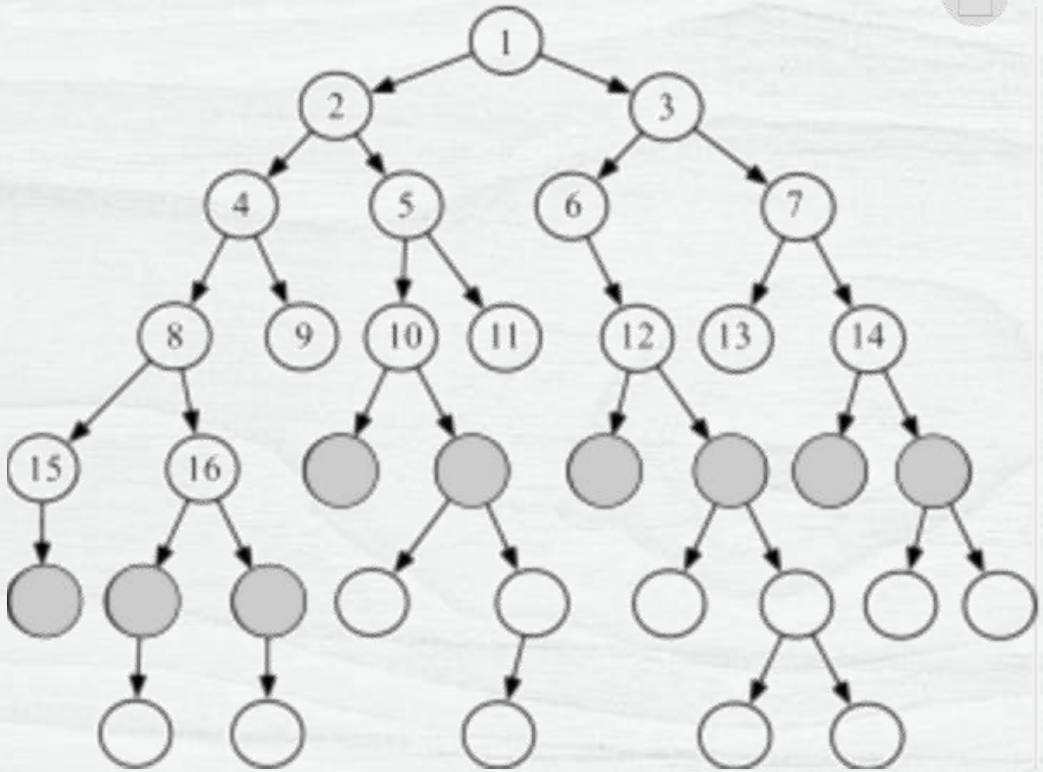
- 完备性：当问题有解时，这个算法能否保证找到解。
- 最优性：搜索策略能否找到最优解。
- 时间复杂度：找到解所需要的时间，也叫搜索代价
- 空间复杂度：执行搜索过程中需要多少内存空间

无信息搜素策略

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)

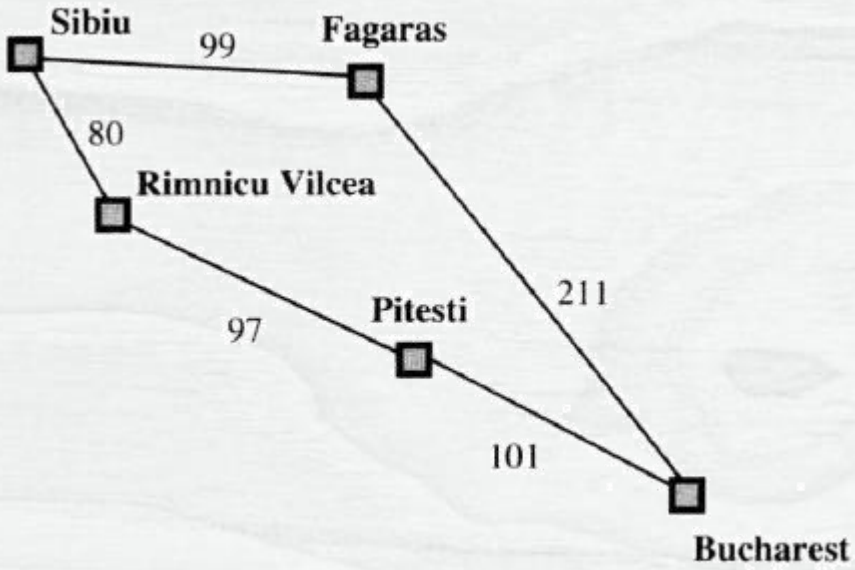


图 3.15　罗马尼亚问题的部分状态空间，用于描述一致代价搜索

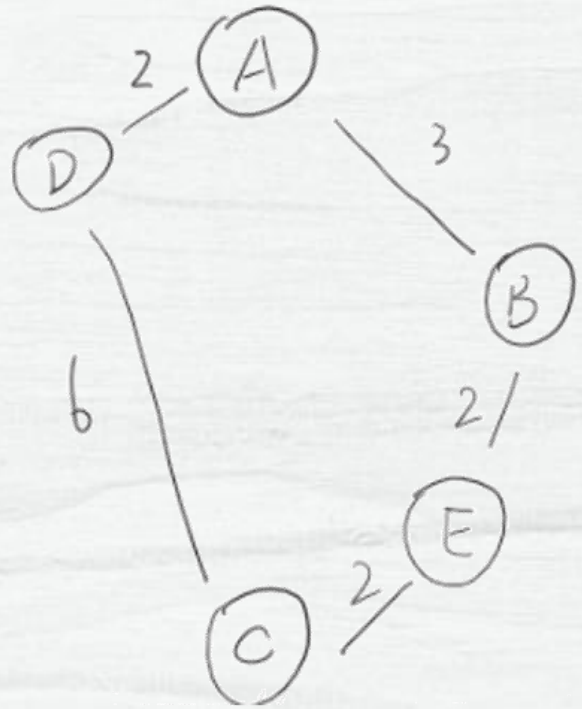
Search Strategy： 扩展最低代价的未扩展节点。

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)

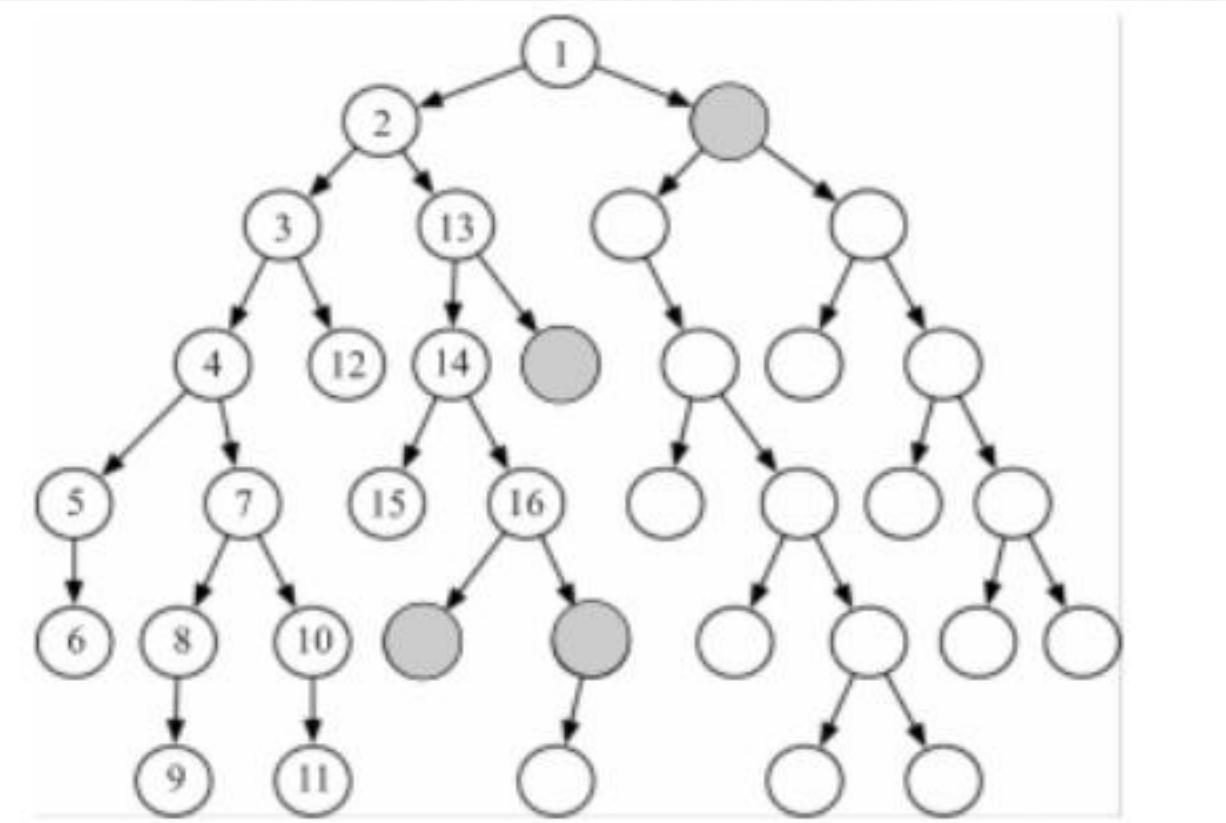


图 3-5　深度优先搜索中节点的扩展顺序

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
```

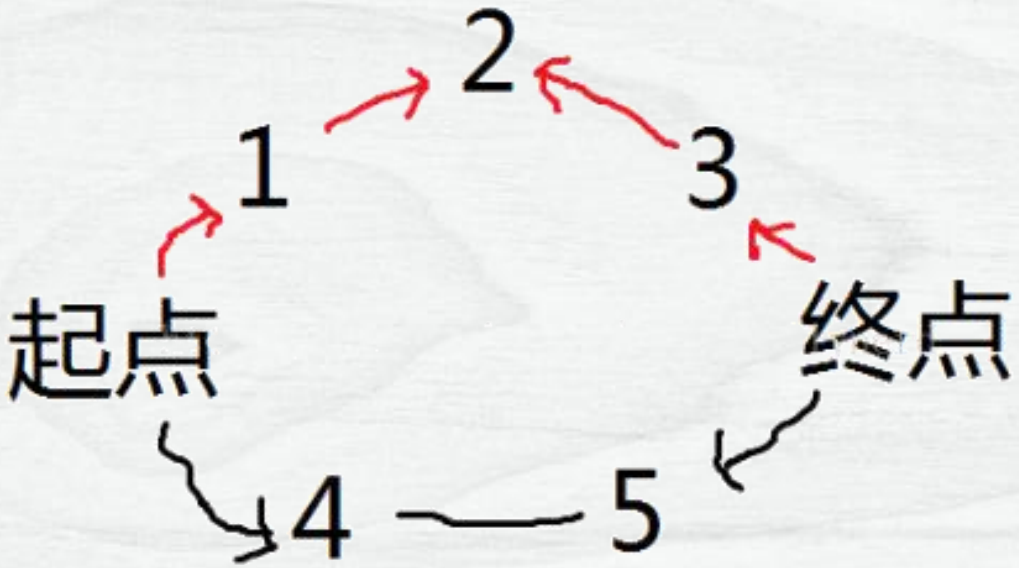
迭代加深搜索（Iterative Deepening Search）

```
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

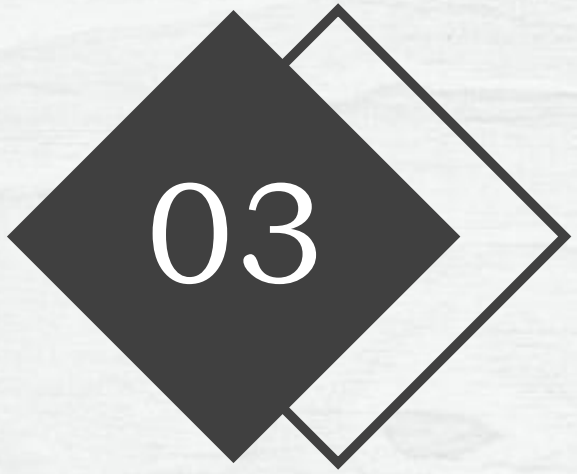
无信息树搜索策略评价

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



启发式搜索策略

启发式搜索

启发式搜索又叫有信息的搜索，它利用问题所拥有的启发信息来引导搜索，达到减少搜索范围，降低问题复杂度的目的。

- 无信息搜索对所有的可能路径节点一视同仁，而启发式搜索可以指导搜索向最有希望的方向前进
- 如何评估一个节点的重要性？ - 估价函数

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

其中g(x)是从初始节点到节点x付出的实际代价；而h(x) 是从节点x到目标节点的最优路径的估计代价。h(x)建模了启发式搜索问题中的启发信息，是算法的关键。启发式函数的设计非常重要，合理的定义才能让搜索算法找到一个最优的问题解。

A* 搜索算法

A*算法可以看作是BFS算法的升级版，在原有的BFS算法的基础上加入了启发式信息。

- A*算法的估价函数也是 $f(x)=h(x)+g(x)$
- 算法描述

从起始节点开始，不断查询周围可到达节点的状态并计算它们的f(x),h(x)与g(x)的值，选取估价函数f(x)最小的节点进行下一步扩展，并同时更新已经被访问过的节点的g(x)，**直到找到目标节点。**

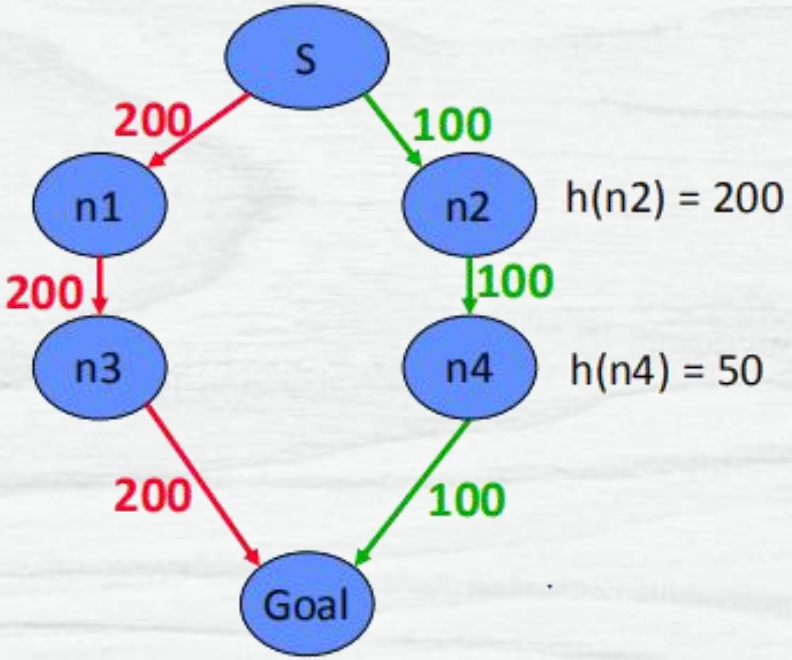
- 算法优缺点：

拥有BFS速度较快的优点，但是因为它要维护“开启列表”以及“关闭列表”，并且需要反复查询状态。因此它的空间复杂度是指数级的。

A* 搜索算法

→ step cost = 200
→ step cost = 100

h(n1) =50
h(n3) =50



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

{S} → {n1 [200+50=**250**], n2 [200+100=**300**]}
→ {n2 [100+200=**300**], n3 [400+50=**450**]}
→ {n4 [200+50=**250**], n3 [400+50=**450**]}
→ {goal [300+0=**300**], n3 [400+50=**450**]}

A* 搜索算法

- 算法步骤

- **开始**

1. 从起点开始，把其当成待处理点存入一个“开启列表”。

2. 搜寻起点周围可能通过的节点，也把它们加入开启列表，为这些节点计算f(x),g(x),h(x)，并且将节点A存为“父节点”。

3. 从开启列表中删除节点A，将其加入“关闭列表”（列表中保存所有不需要再次检查的节点）

- **循环直到找到目标节点或者“开启列表”为空（无路径）**

4. 从“开启列表”中找到估价函数值最低的节点C，并将它从“开启列表”中删除，添加到“关闭列表”中。

5. 检查C所有相邻节点，将其加入“开启列表”，将C作为它们的父节点。

6. 如果新的相邻节点已经在“开启列表”，则更新它们的g(x)值

IDA* （迭代加深A*）搜索算法

IDA* 是迭代加深深度优先搜索算法（IDS）的扩展。因为它不需要去维护表，因此它的空间复杂度远远小于A*。在搜索图为稀疏有向图的时候，它的性能会比A*更好。

- **IDA*算法的估价函数也是** $f(x)=h(x)+g(x)$
- **算法描述**

在算法迭代的每一步，IDA*都进行深度优先搜索，在某一步所有可访问节点对应的最小可估价函数值大于某个给定的阈值的时候，将会剪枝。

- **算法优点**

当问题要求空间复杂度比较低的时候，IDA*更有优势。

IDA* 搜索算法

- 算法步骤

对于给定的阈值bound，定义递归过程

1. 从开始节点C，计算所有邻居节点的估价函数，选取估价函数最小的节点作为下一个访问节点。

2. 对于某个节点，如果估价函数大于阈值，则返回当前节点的估值函数值。

3. 对于某个节点，如果是目标节点，则返回状态“到达”。

启发式函数设计

- 启发式函数h(n)告诉算法从任何节点到目标节点的最小代价估计值，它的选取很大程度影响算法性能。

h(n) 的值	描述	性能变化
h(n) = 0	只有g(n)起作用，退化为Dijkstra算法	保证找到最短路径
h(n) <= h*(n)		保证能找到最短路径
h(n) = h*(n)	只遵循最佳路径不会扩展其它节点	运行速度快并且能找到最短路径
h(n) > h*(n)		不能保证找到最短路径

启发式函数设计

➢性质1：可采纳的（admissible）

当估价函数的预估值小于等于真实值时，算法必然可以找到一条从起始节点到最终节点的最短路径。这种性质叫做相容。

$$h(n) \le h^*(n)$$

➢性质2： 单调的 （consistent）

当节点n的估价函数值永远小于等于它的扩展节点n'的估价函数值时，则启发式函数设计是单调的。

$$h(n) \le \cos t(n, n') + h(n')$$

启发式函数设计

- 不同的应用场景下有很多可选择的启发式函数。比如在网格地图中，一般使用以下几种启发式函数：
- 举个例子：

在正方形网格中，允许向4邻域的移动，使用曼哈顿距离（L1）

在正方形网格中，允许向8邻域的移动，使用对角线距离（L∞）等等

启发函数没有限制，大家可以多尝试几种。

启发式函数设计

- 曼哈顿距离

```
function heuristic(node) =
    // dx
  = abs(node.x - goal.x)
    // dy
  = abs(node.y - goal.y)

     //return
  D * (dx + dy)
```

启发函数可以是曼哈顿距离的D倍。D的设计是为了距离衡量单位与启发函数相匹配。

可以设置为方格间移动的最低代价值。

启发式函数设计

- 对角线距离 （切比雪夫距离）

```
1  function heuristic(node) =
2
3      dx
4   = abs(node.x - goal.x)
5
6      dy
7   = abs(node.y - goal.y)
8
9      return
10  D * max(dx, dy)
```

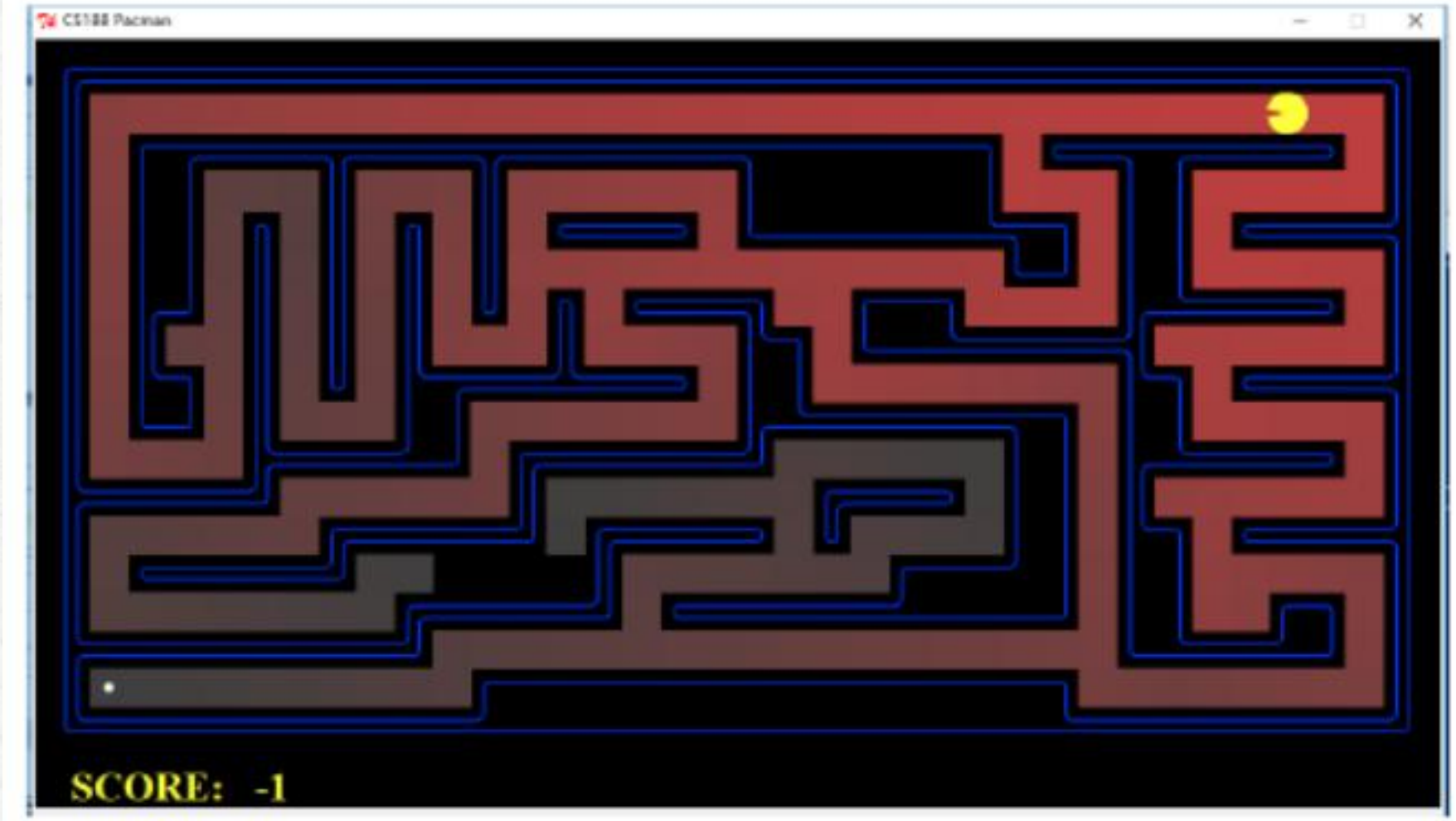
如果在地图中允许朝着对角线方向移动，则可以考虑使用对角线距离。

思考题

- 这些策略的优缺点是什么？它们分别适用于怎样的场景？

实验要求

- 从无信息搜索和启发式搜索算法中分别选择一个策略解决迷宫问题。
  类型一：一致代价、迭代加深、双向搜索
  类型二：A*、IDA*
- 报告要求：报告中①需要有对实现的策略的原理解释，②需要策略的实验效果（四个方面的算法性能）的对比和分析，③思考题回答自己对不同策略优缺点，适用场景的理解和认识 。
- 提交文件
  - 实验报告：17******_wangxiaoming_lab9.pdf。
  - 代码：17******_wangxiaoming_lab9.zip。如果代码分成多个文件，最好写份readme。
- DDL: 2019-11-21 23:00:00


CS188 Pacman
SCORE: -1

```
11111111111111111111111111111111
10000000000000000000000000000001
10111111111111111111111011111101
10110001000100000111111000110001
10110101010101111011111111011011
10110101010100000000000011011001
10110101010101011110011100011101
10100101010001000011011111110001
10110101011111111011000000011011
10110100011000000011111111011001
10000111111011111110000011011101
11111100000010000000111101101001
10000001111110111111010001101101
10111111000000100000001111101001
10000000011111101111111111011001
11111111110000000000000000011101
1E0000000011111111111111100000001
11111111111111111111111111111111
```