人工智能
搜索技术

陈 川
中山大学 计算机学院
2024年

中山大學
SUN YAT-SEN UNIVERSITY

盲目搜索方法回顾

- 宽度优先（Breadth-First）
- 深度优先（Depth-First ）
- 一致代价（Uniform-Cost）
- 深度受限（Depth-Limited）
- 迭代加深搜索（Iterative-Deepening search）

Frontier/边界中的节点选择

- Selection can be achieved by employing an appropriate ordering of the frontier set, i.e.:

  1. Order the elements on the Frontier.
     对边界上的元素进行排序
  2. Always select the first element.
     总是选择第一个元素

- Any selection rule can be achieved by employing an appropriate ordering of the frontier set.

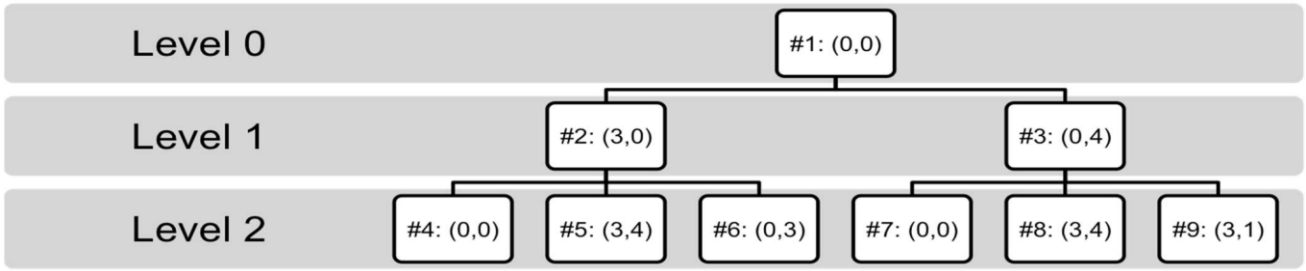
  任何选择规则都可以视为对边界采用某种合适的排序方式

宽度优先搜索

- Place the successors of the current state at the end of the frontier.把当前要扩展的状态的后继状态放在边界的最后
- 例子:
  - 假设使用正整数表示状态{0,1,2,…}
  - 状态n的后继状态为状态n+1和状态n+2
    - E.g. S(1) = {2, 3}; S(10) = {11, 12}
  - 初始状态为0
  - 目标状态为5

宽度优先搜索

{0<>}



{1,2}

{2,2,3}

{2,3,3,4}

{3,3,4,3,4}

{3,4,3,4,4,5}

…

- 假设使用正整数表示状态 {0,1,2,…}
- 状态n的后继状态为状态n+1和状态n+2
  - E.g. S(1) = {2, 3}; S(10) = {11, 12}
- 初始状态为 0
- 目标状态为5

深度优先搜索

Like BFS, but instead of at the back we place the new paths that extend the current path at the front of the Frontier.

把当前要扩展的状态的后继状态放在边界的最前面； 边界上总是扩展最深的那个节点

与宽度优先搜索的示例比较：

{0}

{1,2}

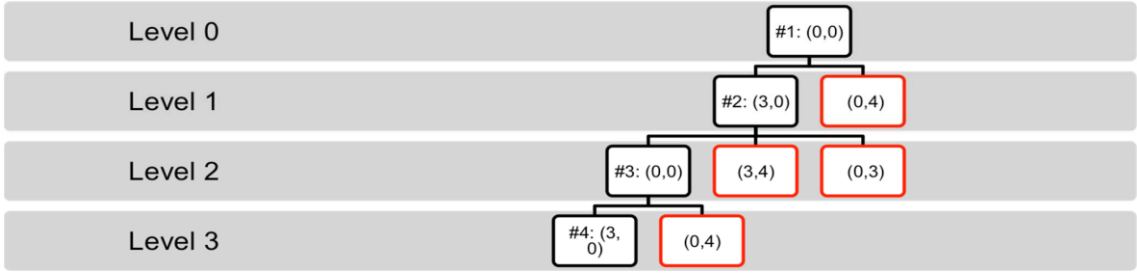
{2,3,2}

{3,4,3,2}

{4,5,4,3,2}

{5,6 4,5,4,3,2}

...



一致代价搜索

- Keep Frontier ordered by increasing cost of the path
  边界中，按路径的成本升序排列
- Always expand the least cost path 总是扩展成本最低的那条路径
- Identical to breadth first if each action has the same cost. 当每种动作的成本是一样的时候，和宽度优先是一样的

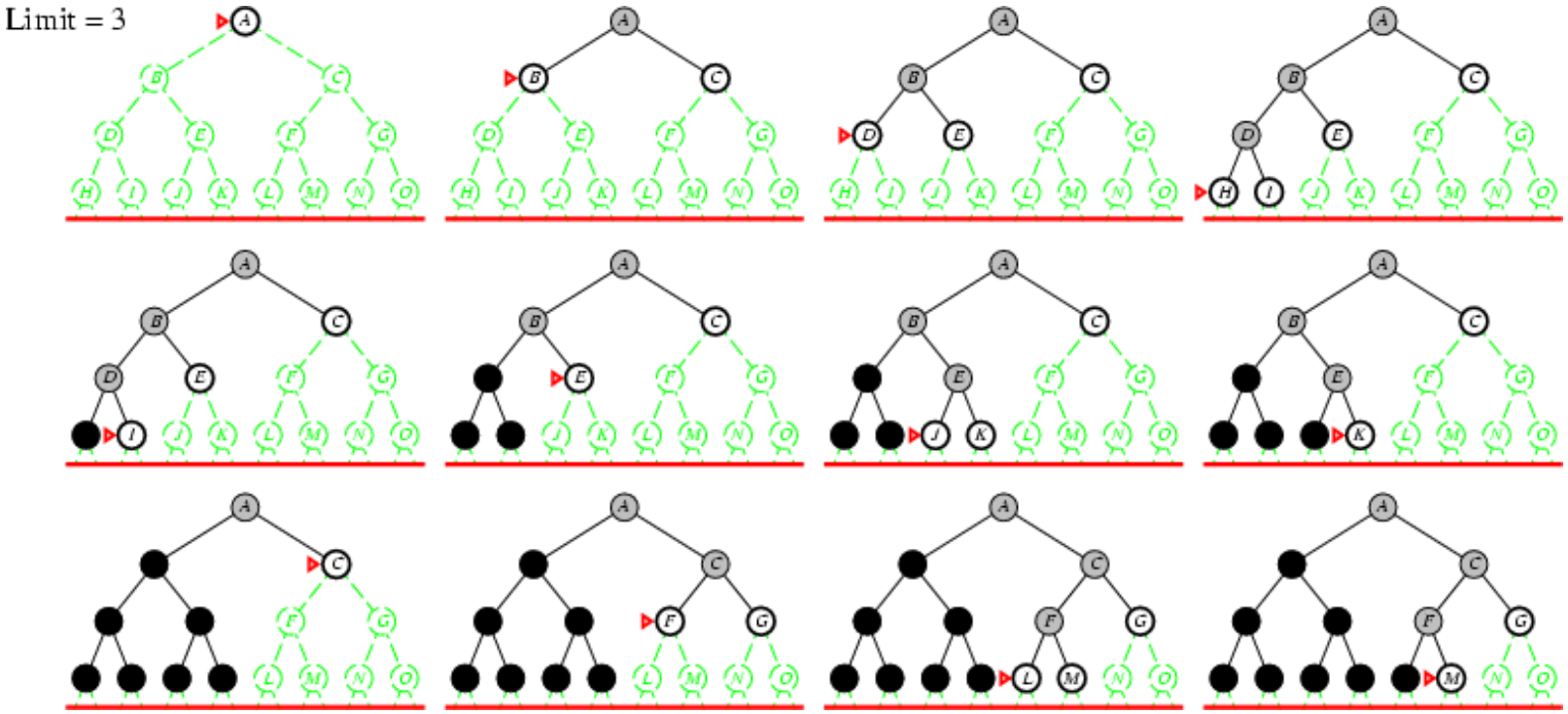
- All cheaper paths are expanded before any more costly path 所有成本较低的路径都会在成本高的路径之前被扩展
- Let C^* be the cost of the optimal solution
- The length of the optimal solution is at most C^*/s
- Hence there are finitely many paths with costs less than C^* 成本小于最优路径的路径数量是有限的

- Eventually we must examine an optimal solution

深度受限搜索

- 宽度优先搜索存在空间复杂度过大的问题
- 深度优先搜索存在可能运行时间非常长，甚至在存在无限路径时无限运行下去的问题
- 深度受限搜索
  - 深度优先搜索，但是预先限制了搜索的深度 L
  - 因此无限长度的路径不会导致深度优先搜索无法停止的问题
  - 但只有当解路径的长度 $\leq L$ 时，才能找到解

深度受限搜索



迭代加深搜索

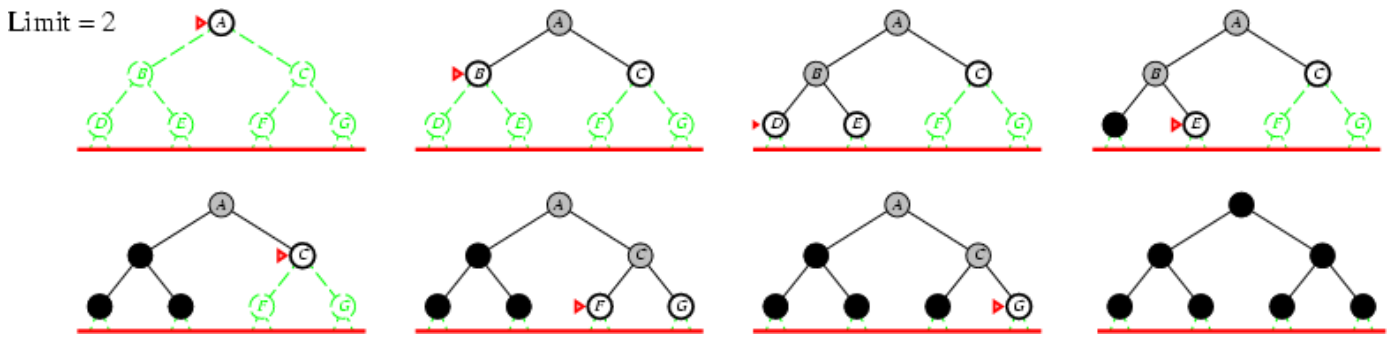
- 为了解决深度优先搜索和宽度优先搜索存在的问题
- 一开始设置深度限制为$L = 0$，我们迭代地增加深度限制，对于每个深度限制都进行深度受限搜索
- 如果找到解，或者深度受限搜索没有节点可以扩展的时候可以停止当前迭代，并提高深度限制L
- 如果没有节点可以被剪掉（深度限制不能再提高）仍然没有找到解，那么说明已经搜索所有路径，因此这个搜索不存在解

Iterative Deepening Search

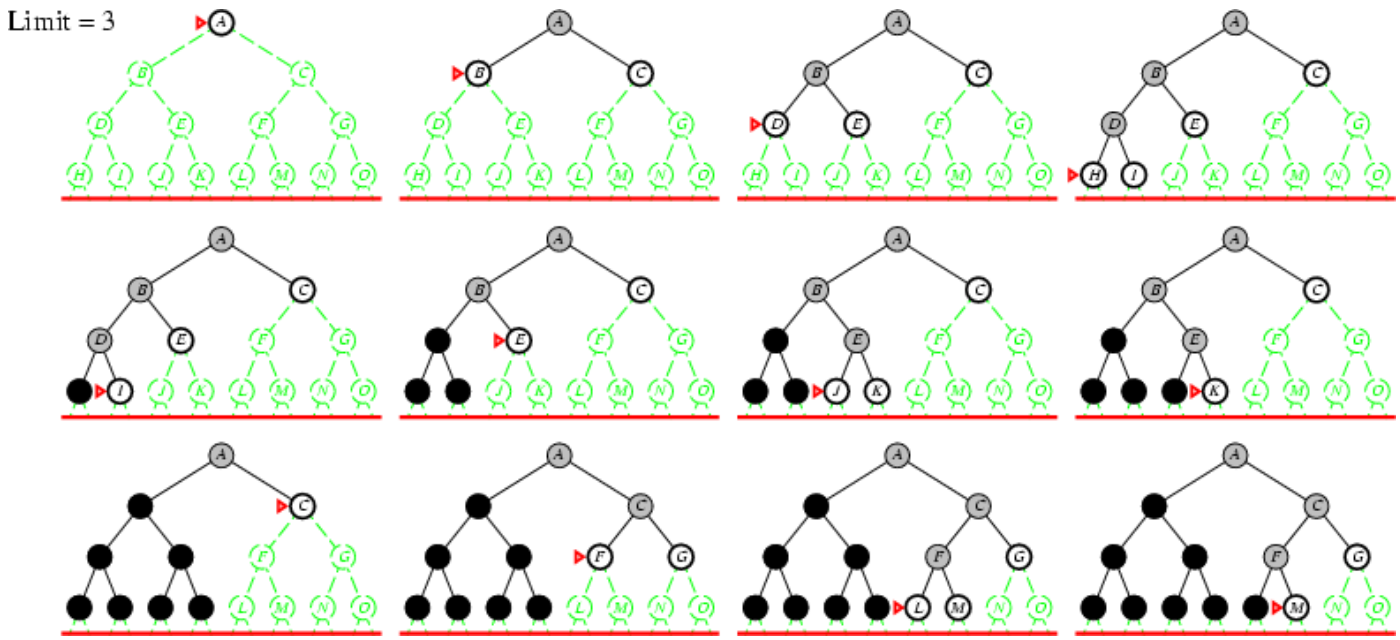
Limit = 0

Iterative Deepening Search

Iterative Deepening Search



Iterative Deepening Search



无信息搜索总结

- 宽度优先：搜索对象的位置深度
- 一致代价：搜索对象的到达路径长度
- 深度优先：搜索对象的位置深度
- 深度受限：搜索对象的位置深度
- 迭代加深：搜索对象的位置深度
- 双向：搜索对象的位置深度

共有的特征是：

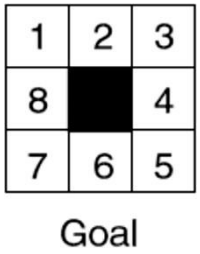
- 搜索方向都依据了某一评价指标
- 搜索方向和搜索对象本身的属性无关

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
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]

如何能让搜索更“聪明”？

- 通用搜索策略在搜索过程中，不对状态优劣进行判断，仅按照固定方式搜索。在盲目搜索中，我们没有考虑边界上的节点哪一个更具有“前景”（promising）
- 例如在一致代价搜索(UCS)时，我们总是扩展从初始状态到达当前状态的成本最小的那条路径，却没有考虑过从当前状态点沿着当前路径到达目标路径的成本

人解决问题的“启发性”



对两个可能的状态A和B，选择“从目前状态到最终状态”更好的一个作为搜索方向。

- 但很多时候我们对两个状态的优劣是有判断的。

搜索 Search

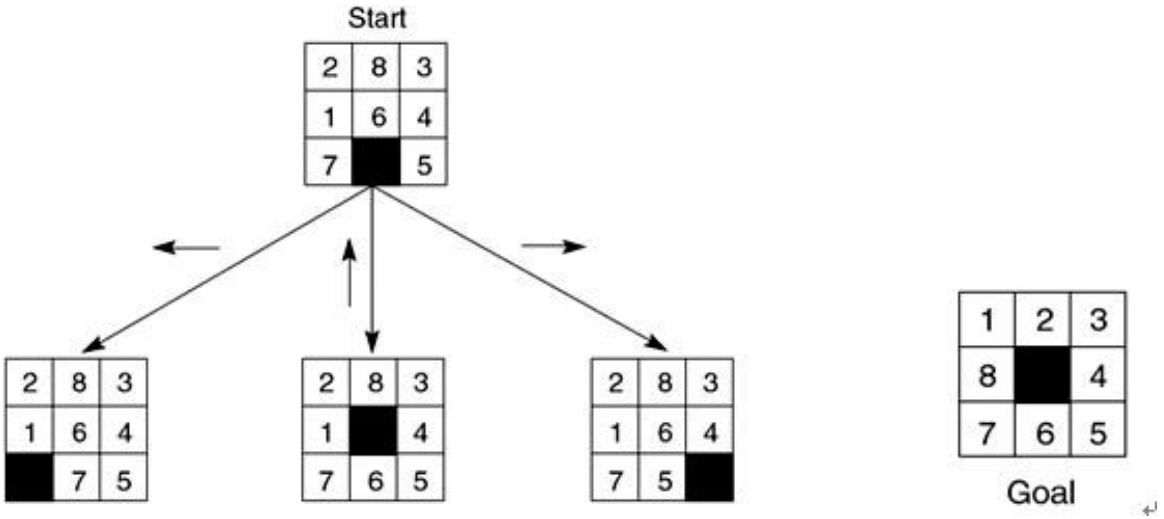
- Problem solving by search 搜索可以解决的问题
- Uninformed search 盲目（无信息）搜索
- Heuristic search 启发式（有信息）搜索
  - A* 搜索
  - A* 的性质
  - 如何构造启发式函数
  - 在实例中运行A*

启发式搜索

- 在许多情况下，我们可以有额外的知识来衡量当前节点，例如可以知道当前节点到达目标节点的成本
- 对于一个具体的问题，构造一个专用于该领域的启发式函数$h(n)$, 该函数用于估计从节点n到达目标节点的成本/当前状态与目标状态的接近程度.
- 要求对于所有满足目标条件的节点n，$h(n) = 0$
- 在不同的问题领域中，启发式函数是随领域不同而不同的, 通常可以是
  - 当前节点到目标的某种距离或者差异的度量；
  - 当前节点处于最佳路径的概率；
  - 某种条件下的主观if-then规则；

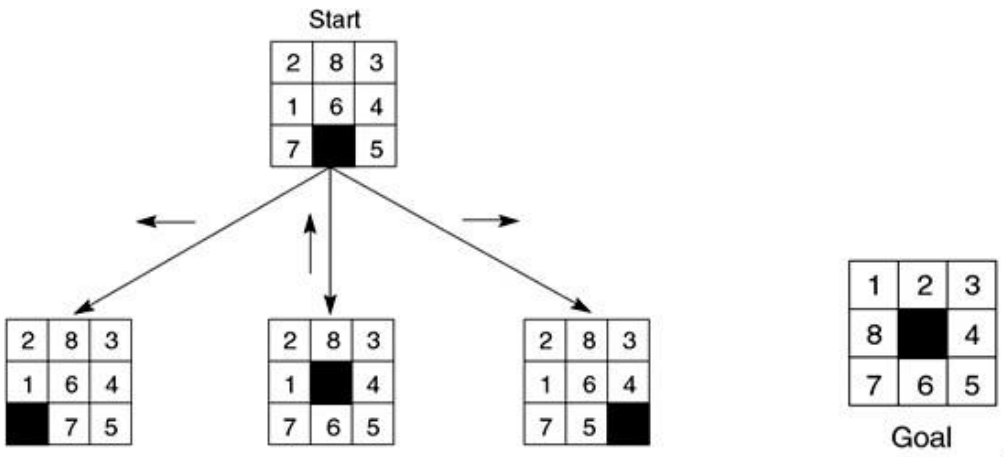
启发式函数示例：8-puzzle

- 在某个状态下，共有三种可能的选择。
- 如何评判三种走法的优劣？



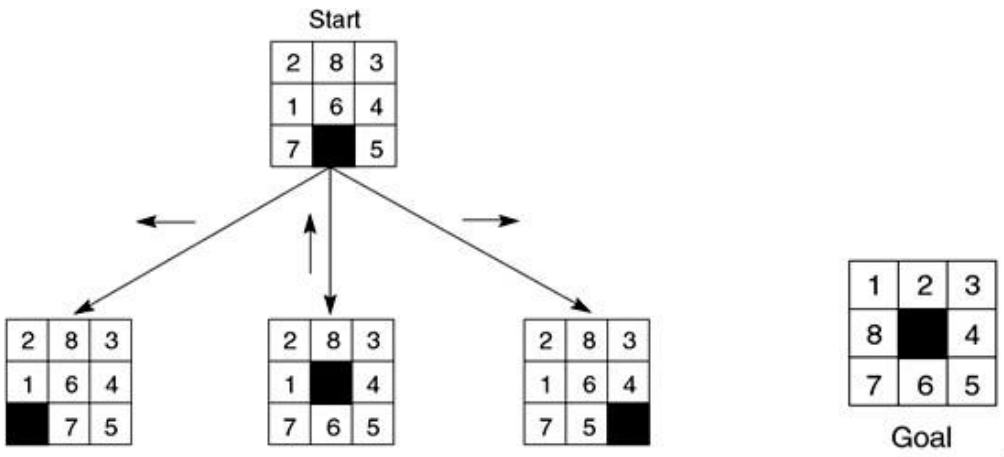
启发式函数示例：8-puzzle

- **Method A：**
  - 当前棋局与目标棋局之间错位的牌的数量，错数最少者最优。
  - 然而，这个启发方法没有考虑到距离因素，譬如：棋局中把“1”“2”颠 倒，与“1”“5”颠倒，但是移动难度显然不同。



启发式函数示例：8-puzzle

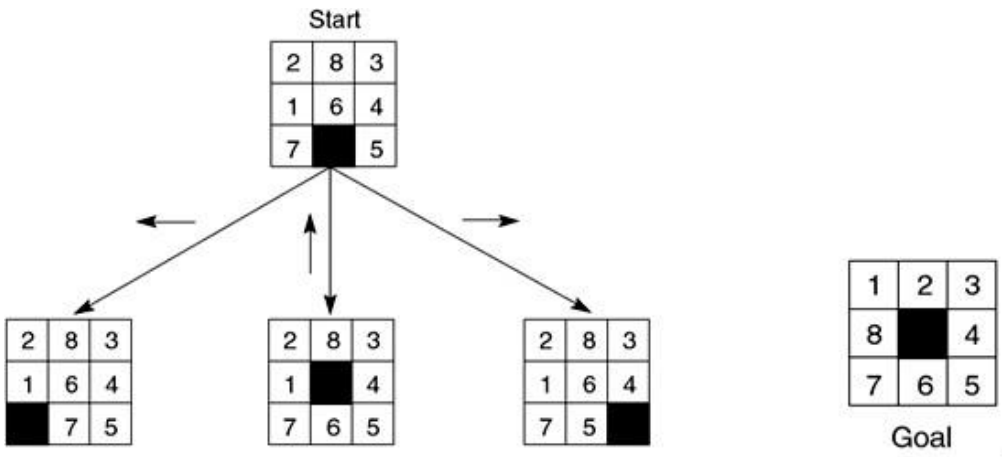
- **Method B改进：**
  - 更好的启发方法是“错位的牌距离目标位置的距离和最小”。
  - Method B 仍然存在很大的问题：没有考虑到牌移动的难度。两张牌即使相差一格，如“1”“2”颠倒，将其移动至目标状态依然不容易。



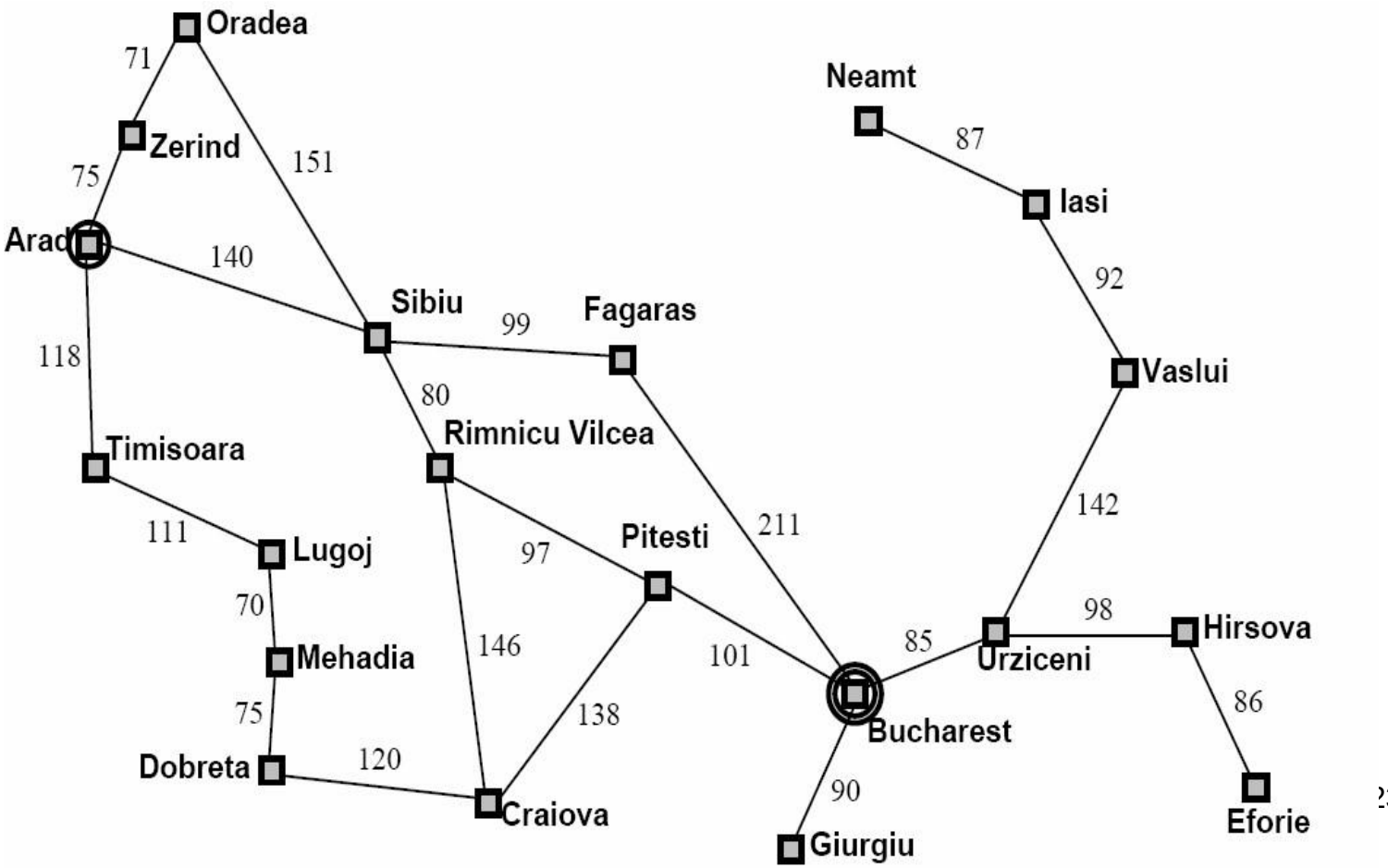
启发式函数示例：8-puzzle

- **Method C：**
  - 在遇到需要颠倒两张相邻牌的时候，认为其需要的步数为一个固定的数字。

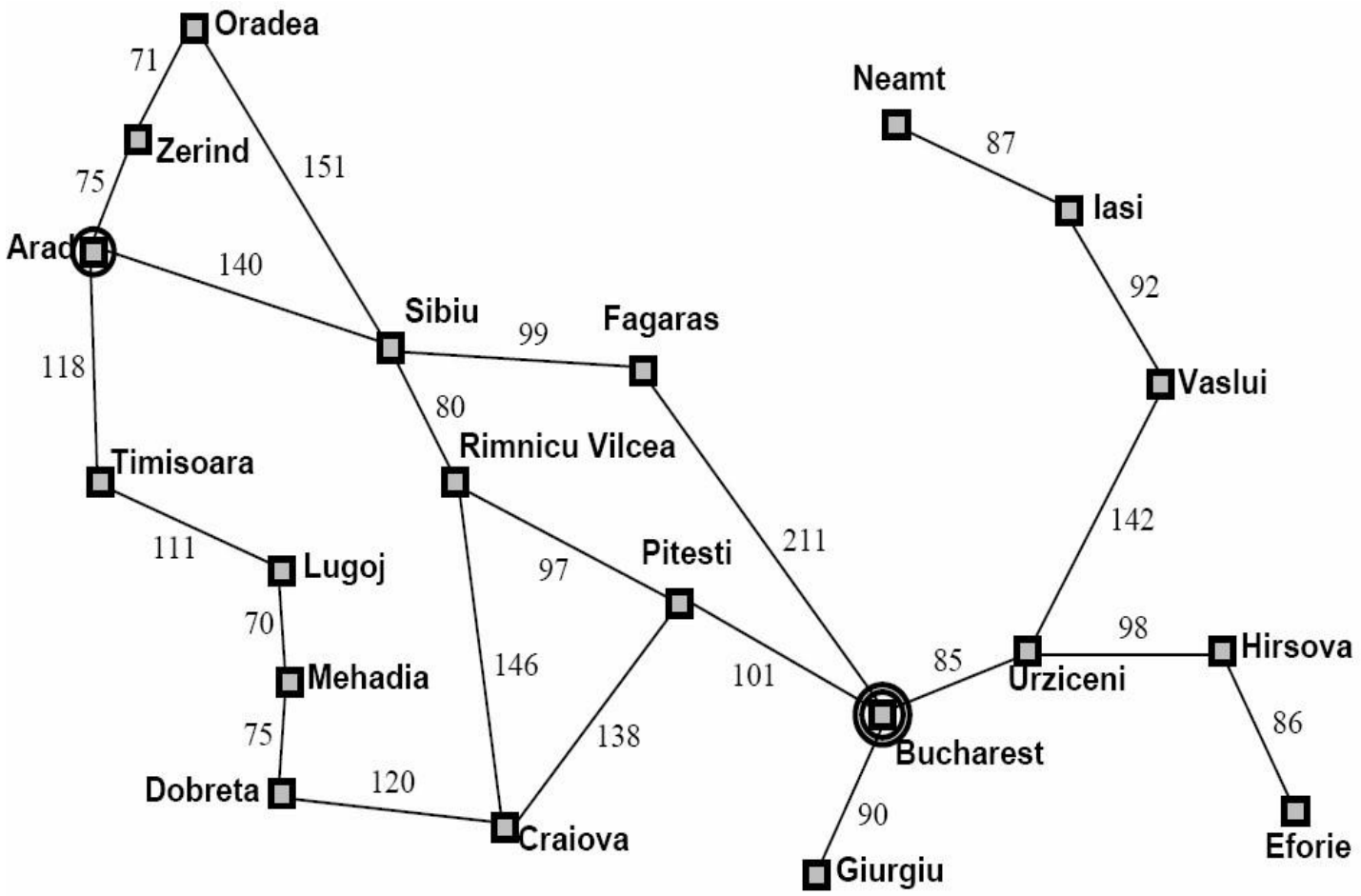
- **Method D改进：**
  - 将B与C的组合，考虑距离，同时再加上需要颠倒的数量。



启发式函数示例：直线距离（欧氏距离）



启发式函数示例：直线距离（欧氏距离）



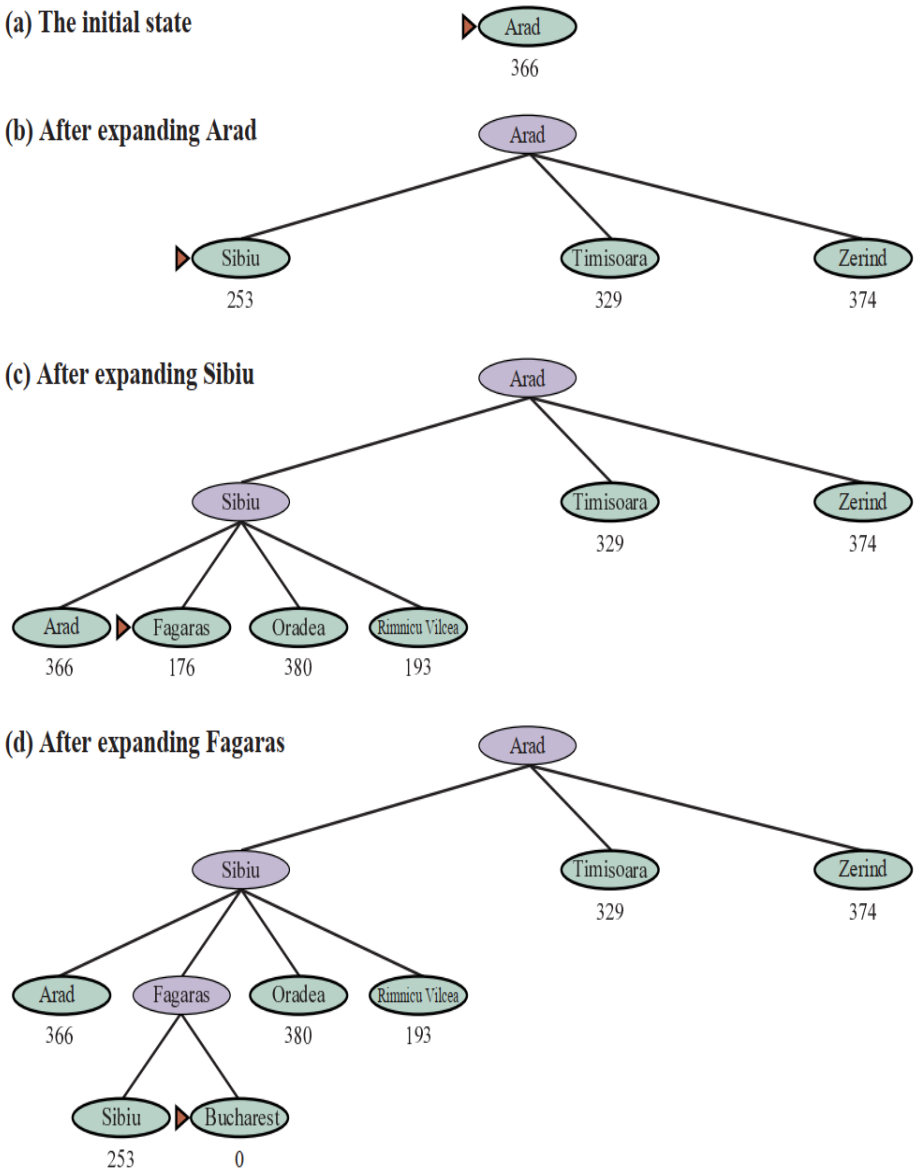
Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

贪心最佳优先搜索(Greedy BFS)

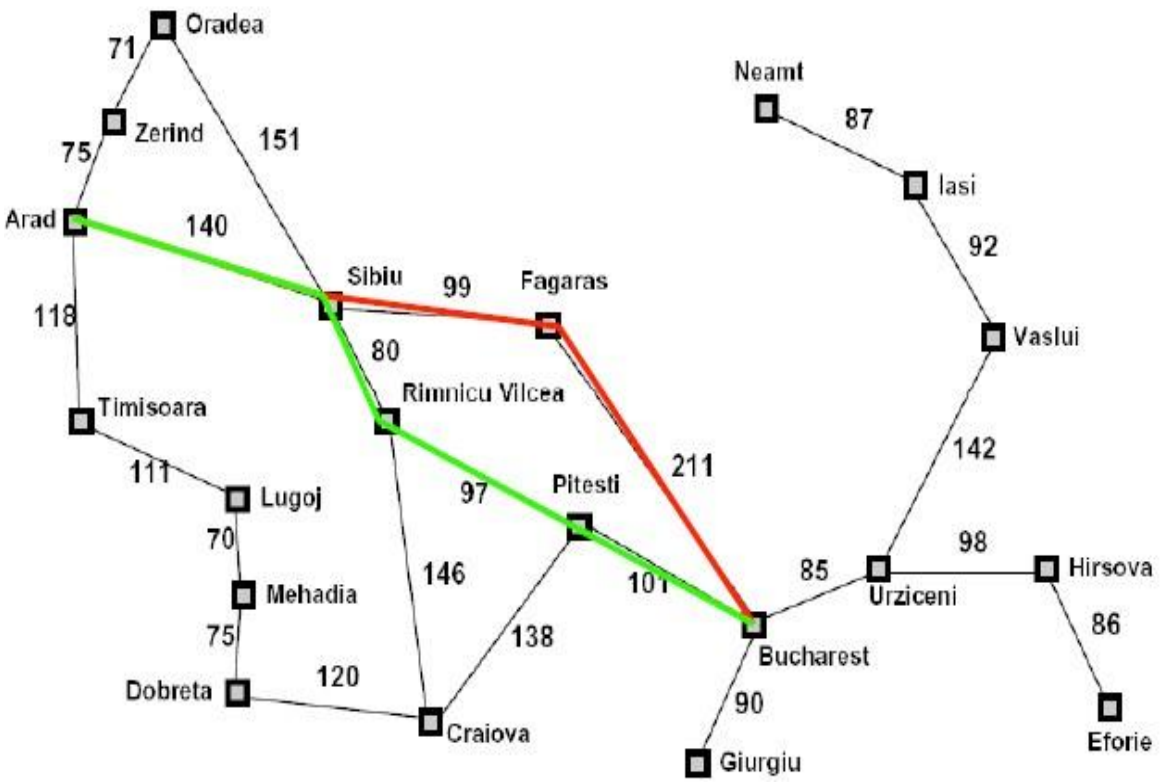
- 利用启发式函数$h(n)$来对边界上的节点进行排序
- 我们贪心地希望找到成本最低的解，这种只考虑启发式函数进行搜索的方式叫做贪心最佳优先搜索(Greedy Best-first Search)

示例

Straight−line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374



示例



Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

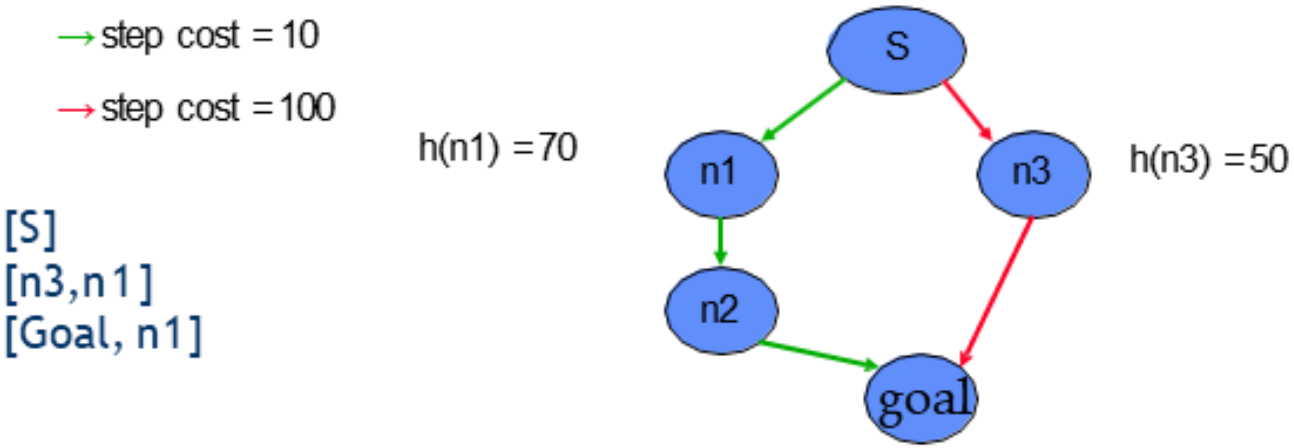
Arad-Sibiu-Fagaras-Bucharest: 140 + 99 + 211 = 140 + 310 = 450

Arad-Sibiu-RV-Pitesli-Bucharest: 140 + 80 + 97 + 101 = 140 + 278 = 418

In red is the path we selected. In green is the shortest path between Arad and Bucharest. What happened?

贪心最佳优先搜索(Greedy BFS)

- 但是，这种做法忽略了从初始状态到达节点n的成本
- 因此这种做法可能“误入歧途”，选择了离初始状态很远（成本很高），但根据$h(n)$看起来离目标状态很近的节点

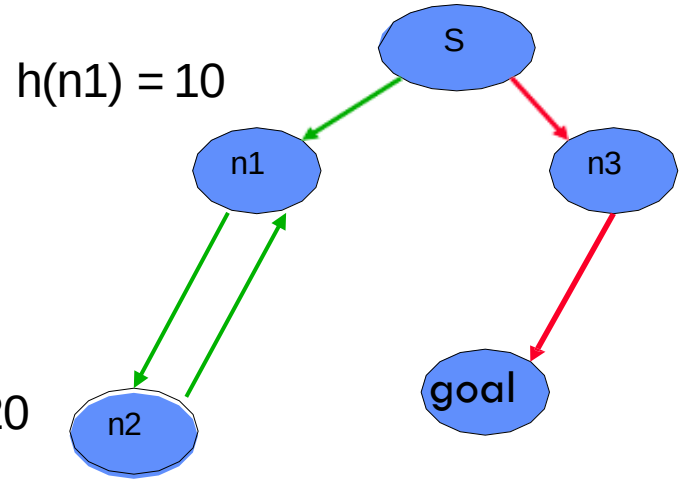


因此Greedy BFS 不是最优的。

贪心最佳优先搜索(Greedy BFS)

→ step cost = 10

→ step cost = 100



{S}
{n1,n3}
{n2,n3}
{n1,n3}
…

因此Greedy BFS 也不是完备的

贪心最佳优先搜索(Greedy BFS)

单纯依靠启发函数搜索不可行

- 对于一个具体问题，我们可以定义最优路线：“从初始节点出发，以最优路线经过当前节点，并以最优路线达到终止节点”。
  - 盲目搜索，只考虑了前半部分，能计算出从初始节点走到当前节点的优劣。
  - 启发函数则只“估计”了当前节点到最终节点的优劣。

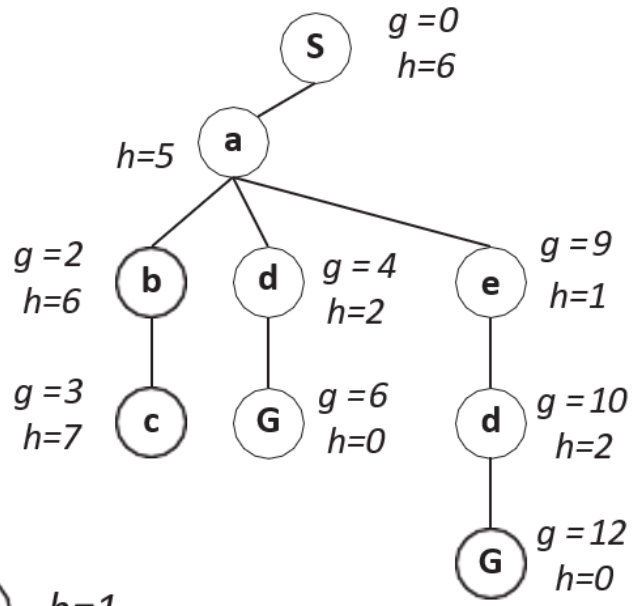
- 两者相结合，就是启发式搜索策略。
- 典型代表是A算法。

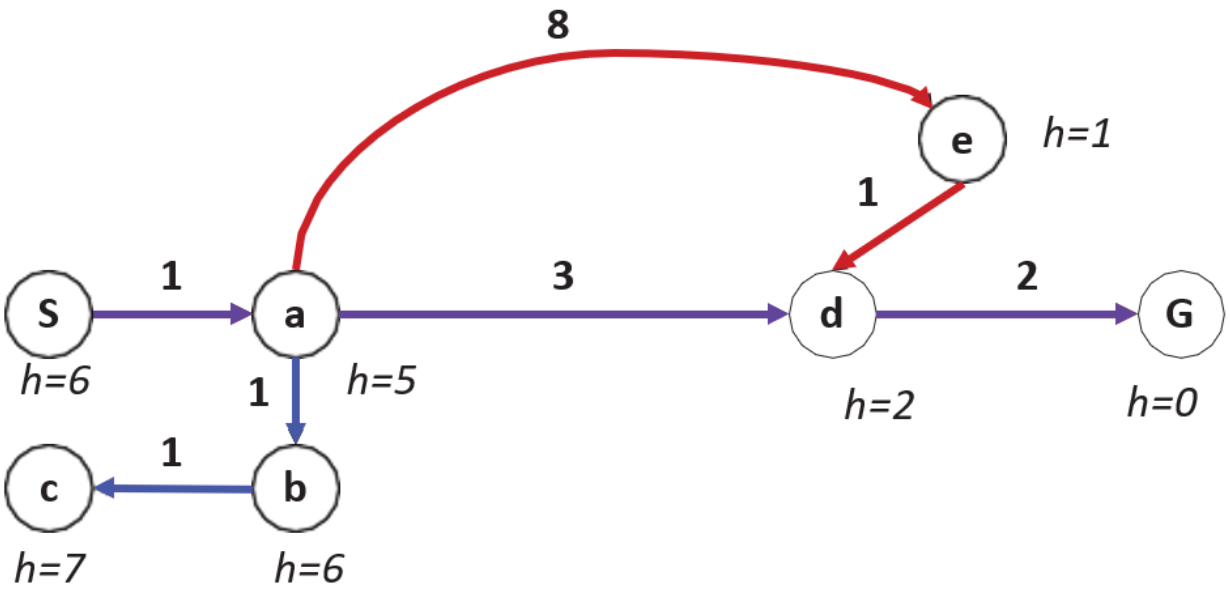
A搜索

- Define an evaluation function 定义评价函数 $f(n) = g(n) + h(n)$
  - $g(n)$ is the cost of the path to node n
    从初始节点到达节点n的路径成本
  - $h(n)$ is the heuristic estimate of the cost of getting to a goal node from n
    从n节点到达目标节点的成本的启发式估计值
- So $f(n)$ is an estimate of the cost of getting to the goal via node n. 是经过节点n从初始节点到达目标节点的路径成本的估计值
- We use $f(n)$ to order the nodes on the frontier. Always expand the node with lowest f-value on Frontier. 利用节点对应的$f(n)$值来对边界上的节点进行排序，并总扩展边界中具有最小f值的节点。
- 对于某个确定状态，g(n)和h(n)都是定值，用两者的和评估当前节点到达最终目标的成本，采用最佳优先搜索进行求解

一致代价搜索UCS和贪婪搜索Greedy对比

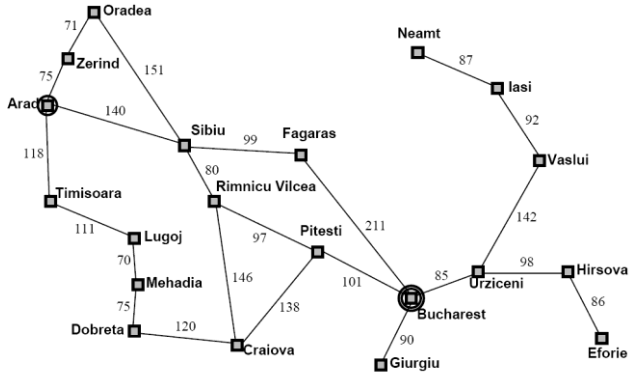
- UCS：按路径计算成本
  - 路径代价g(x)
- Greedy：按目标临近性计算成本
  - 启发函数h(x)





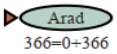
A*使用二者之和f (x) = g(x) + h(x)

示例

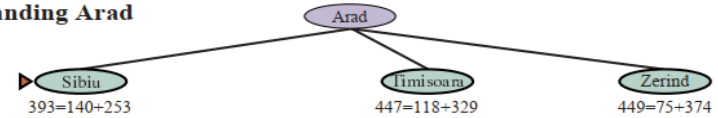


Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	**100**
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

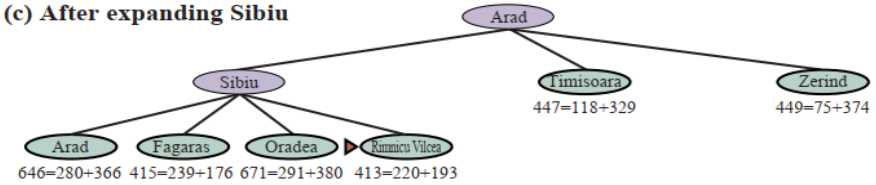
(a) The initial state



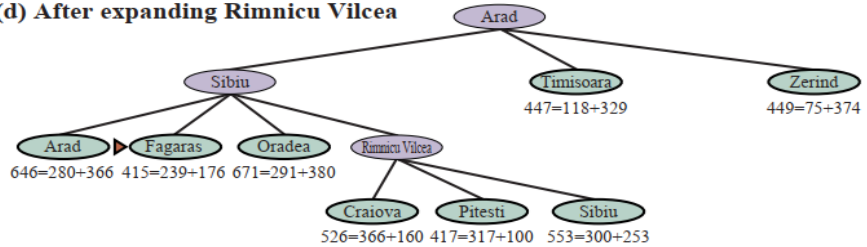
(b) After expanding Arad



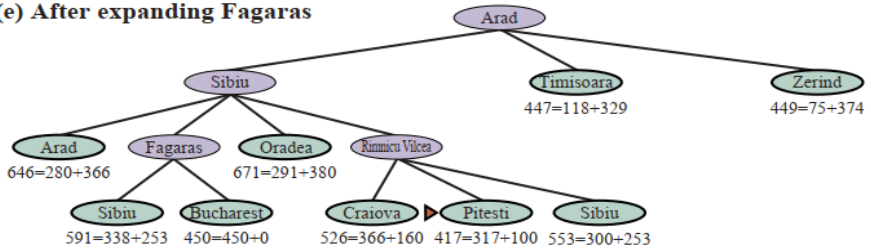
(c) After expanding Sibiu



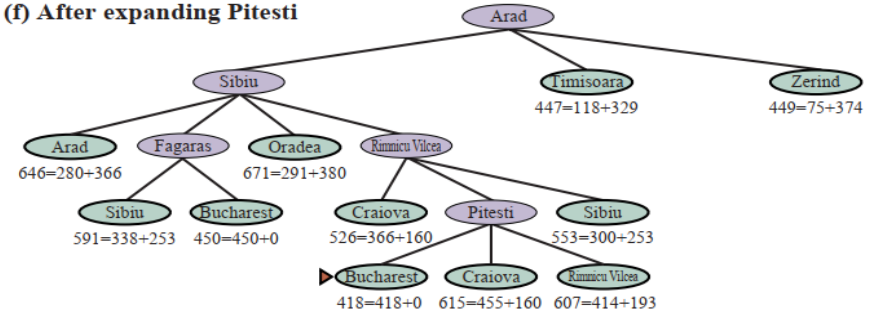
(d) After expanding Rimnicu Vilcea



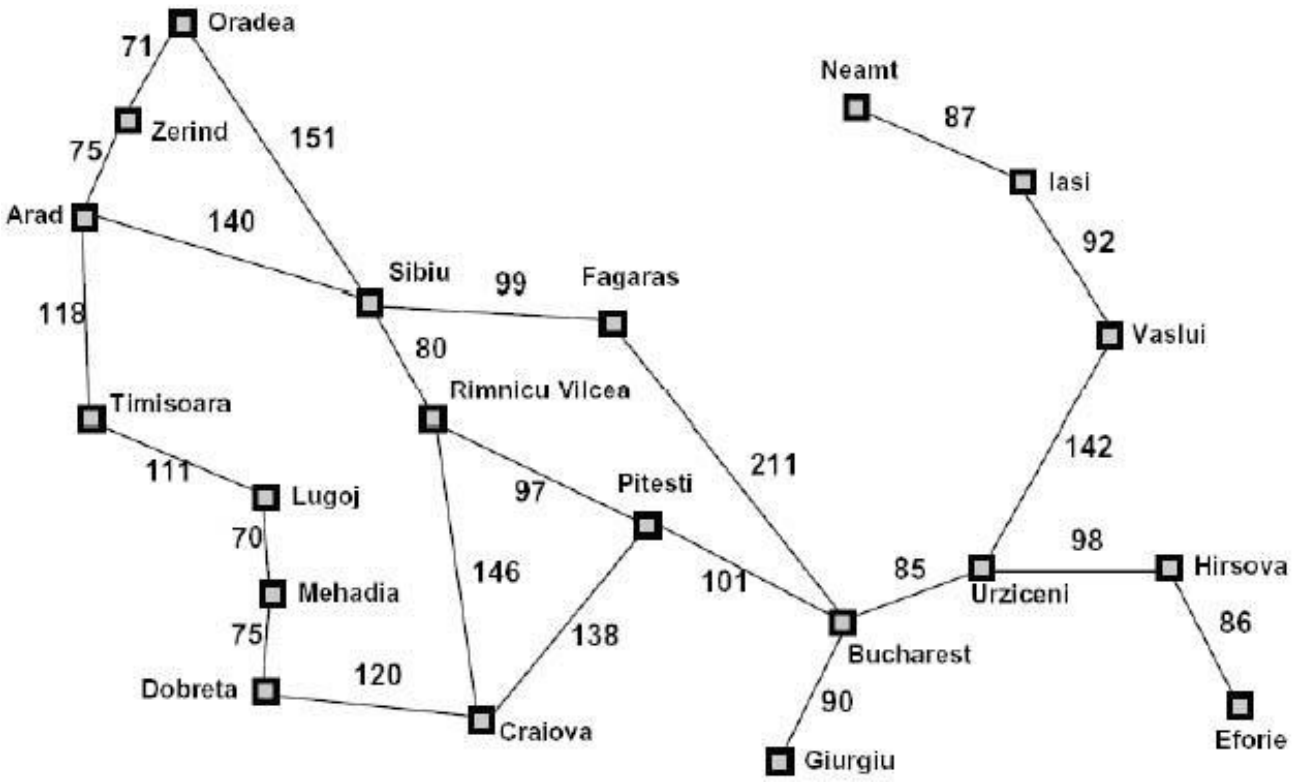
(e) After expanding Fagaras



(f) After expanding Pitesti



示例



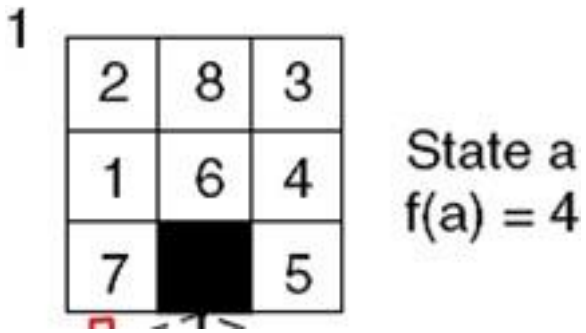
Straight−line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Arad-Sibiu-RV-Pitesli-Bucharest: 418

示例 8-puzzle的启发式求解

- 仍然以8格拼图来解释A算法。令h(n)为“错牌数量”，g(n)为“已经移动的步数”，从初始状态开始搜索。

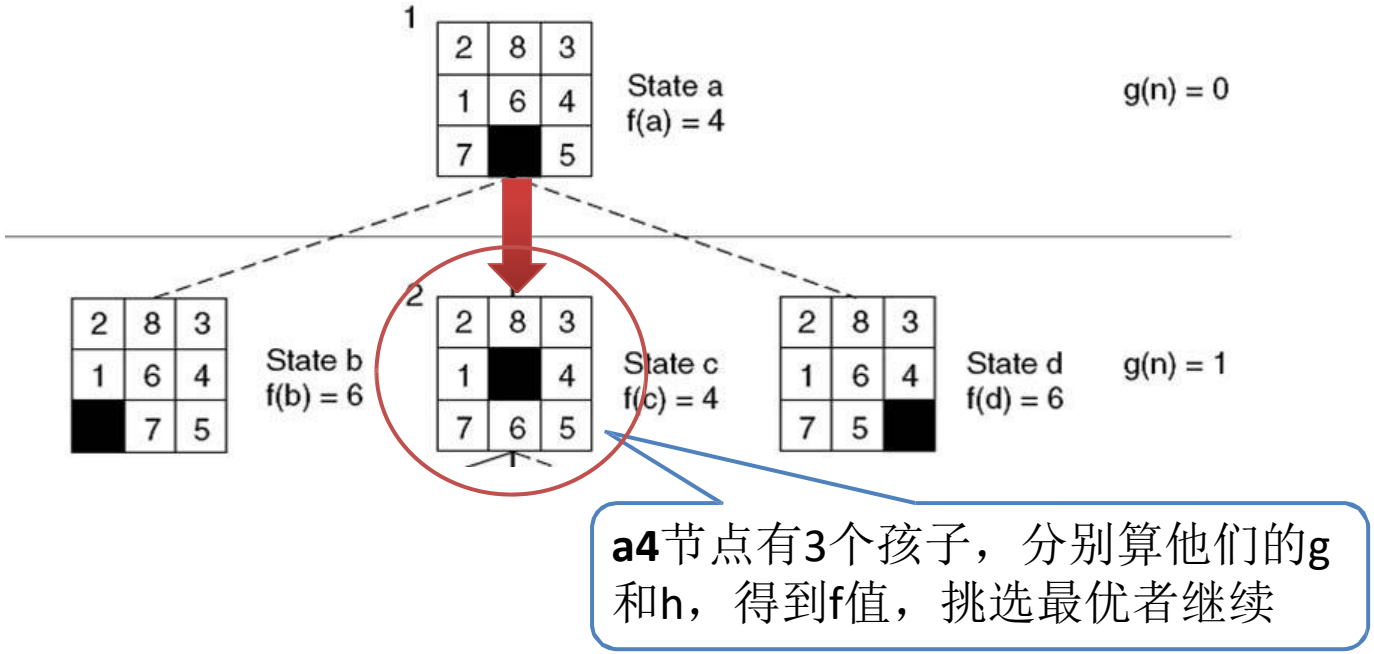
- step1：



最初，节点g=0，h=4。命名该节点为a4

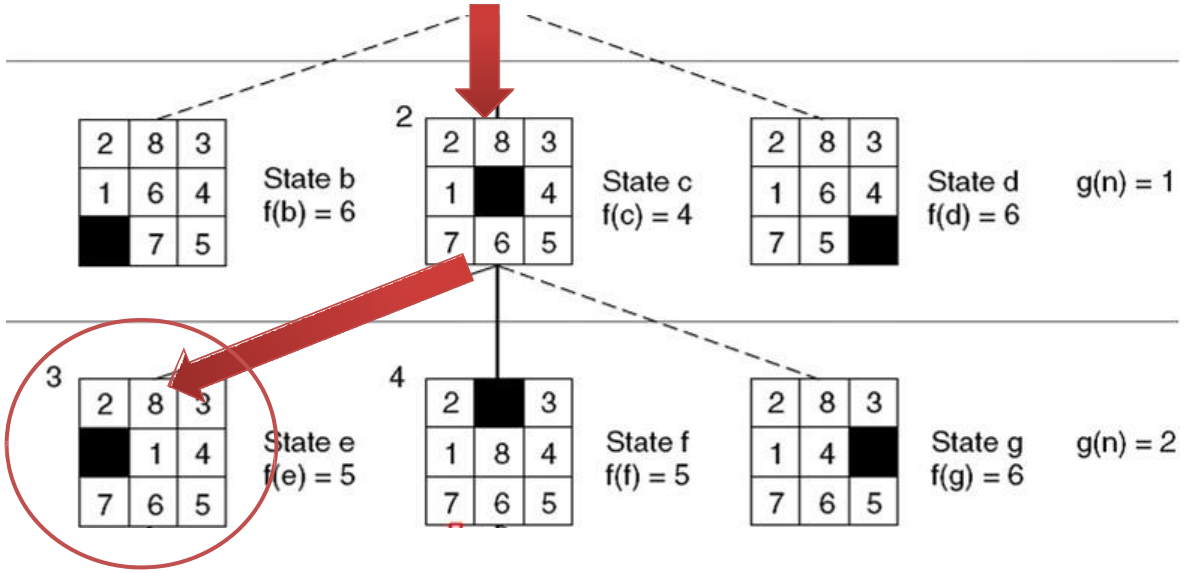
Open =[**a4**]
Closed=[ ]

- Step2：三种走法



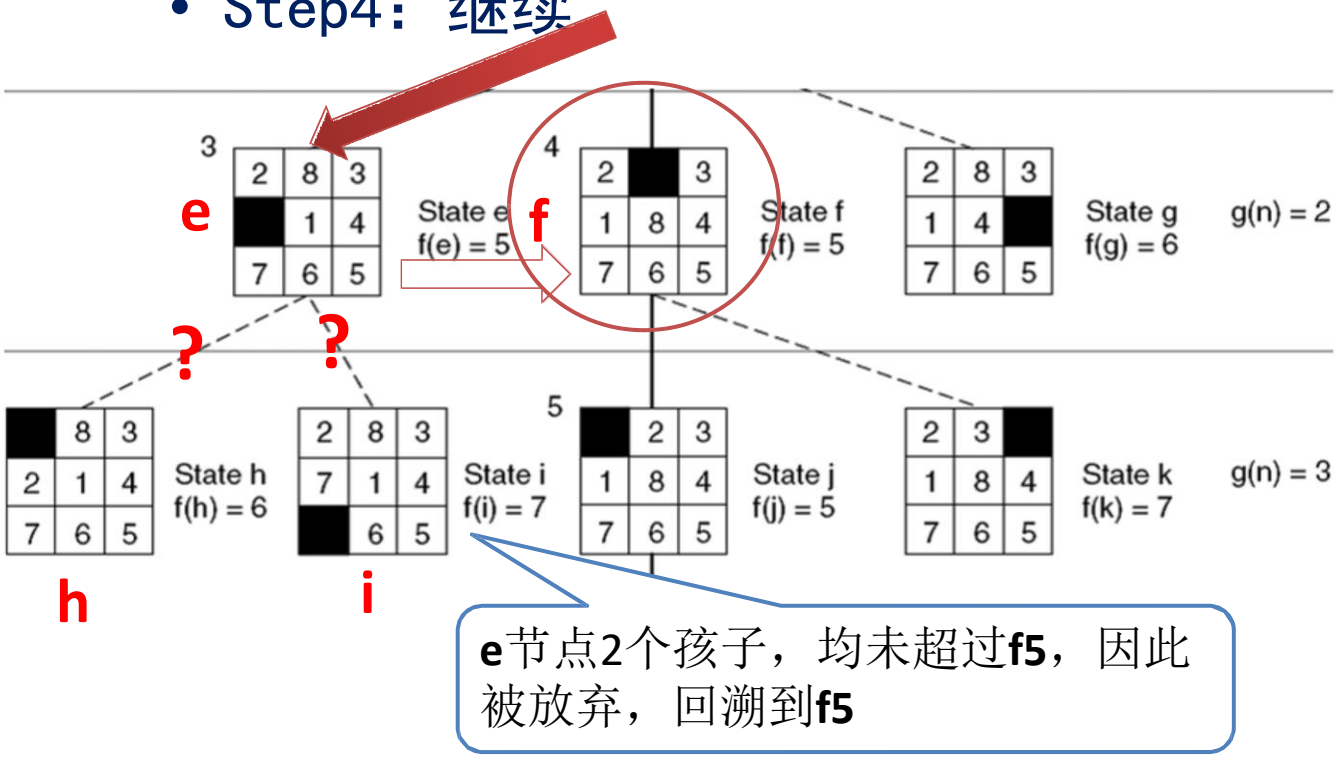
Open =[**c4,b6,d6**]
closed=[**a4**]

- Step3：继续



Open =[**e5,f5**,**b6,d6,g6**]
closed=[**a4,c4**]

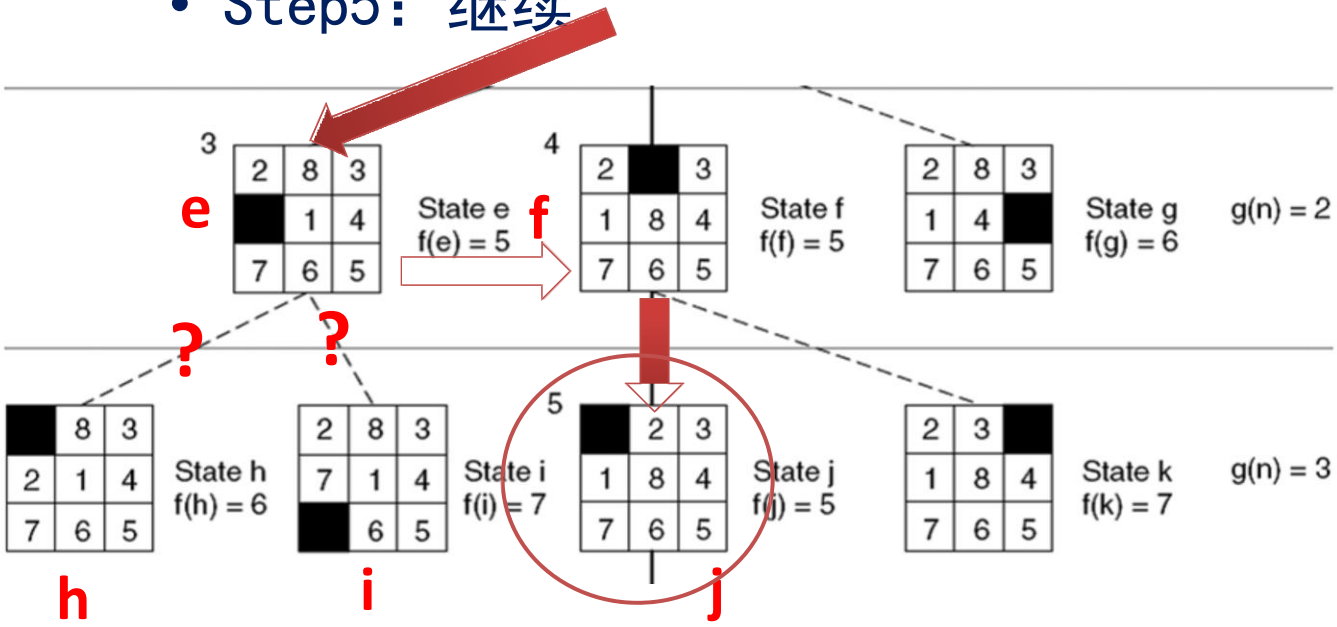
- Step4：继续



e节点2个孩子，均未超过**f5**，因此被放弃，回溯到**f5**

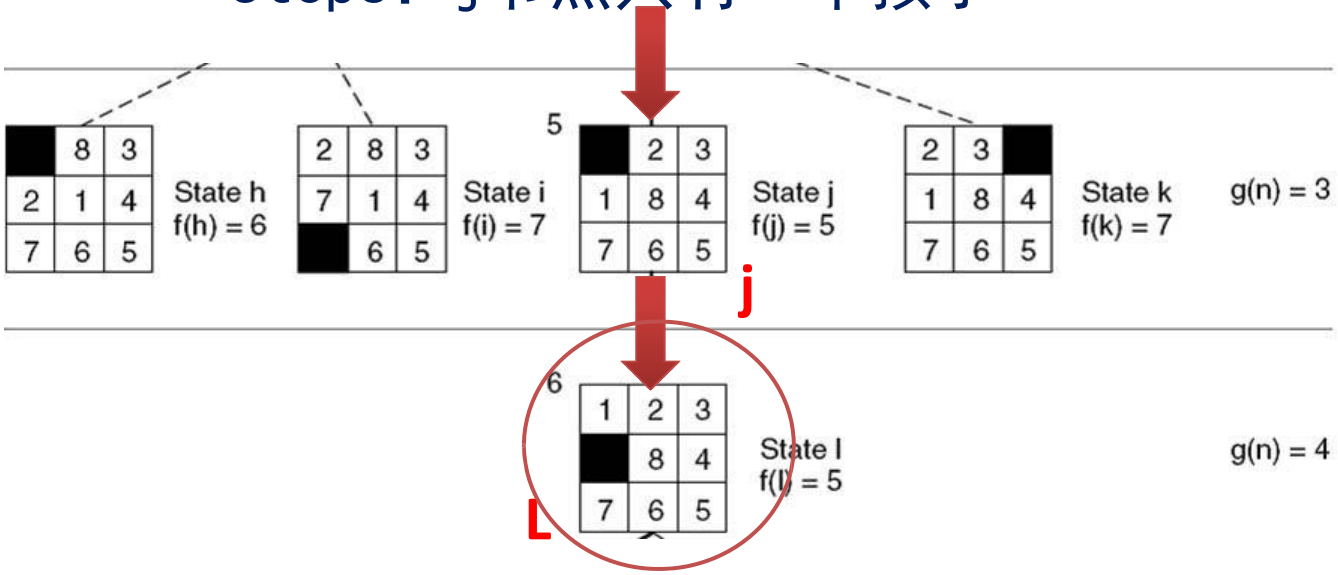
Open =[**f5**,**h6**,**b6**,**d6**,**g6**,**i7**]
closed=[**a4**,**c4**,**e5**]

- Step5：继续



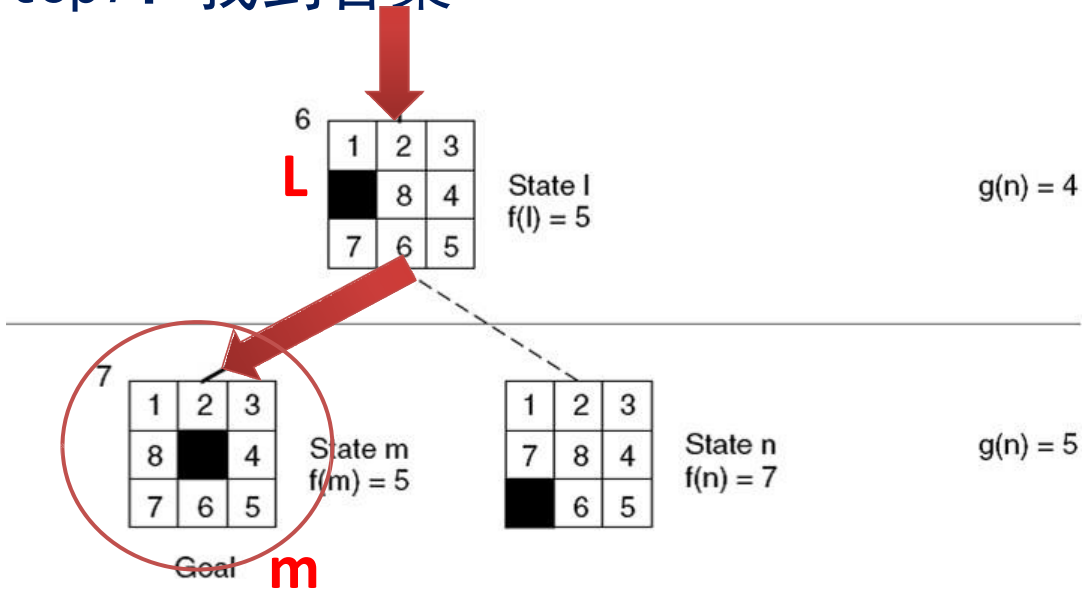
Open =[**j5**,**h6**,**b6**,**d6**,**g6**,**k7**,**i7**]
closed=[**a4**,**c4**,**e5**,**f5**]

- Step6：j节点只有一个孩子



Open =[**L5,h6,b6,d6,g6,k7,i7**]
closed=[**a4,c4,e5,f5,j5**]

- Step7：找到答案



Open =[**m5**,**h6**,**b6**,**d6**,**g6**,**n7**,**k7**,**i7**]
closed=[**a4**,**c4**,**e5**,**f5**,**j5**,**L5**]

示例与代码：https://www.redblobgames.com/pathfinding/a-star/introduction.html

Uniform Cost Search

```
frontier = PriorityQueue()
frontier.put(start, 0)
came_from = dict()
cost_so_far = dict()
came_from[start] = None
cost_so_far[start] = 0

while not frontier.empty():
   current = frontier.get()

   if current == goal:
      break

   for next in graph.neighbors(current):
      new_cost = cost_so_far[current] + graph.cost(current, next)
      if next not in cost_so_far or new_cost < cost_so_far[next]:
         cost_so_far[next] = new_cost
         priority = new_cost
         frontier.put(next, priority)
         came_from[next] = current
```

Greedy Best-First Search

```
frontier = PriorityQueue()
frontier.put(start, 0)
came_from = dict()
came_from[start] = None

while not frontier.empty():
   current = frontier.get()

   if current == goal:
      break

   for next in graph.neighbors(current):
      if next not in came_from:
         priority = heuristic(goal, next)
         frontier.put(next, priority)
         came_from[next] = current
```

A Search

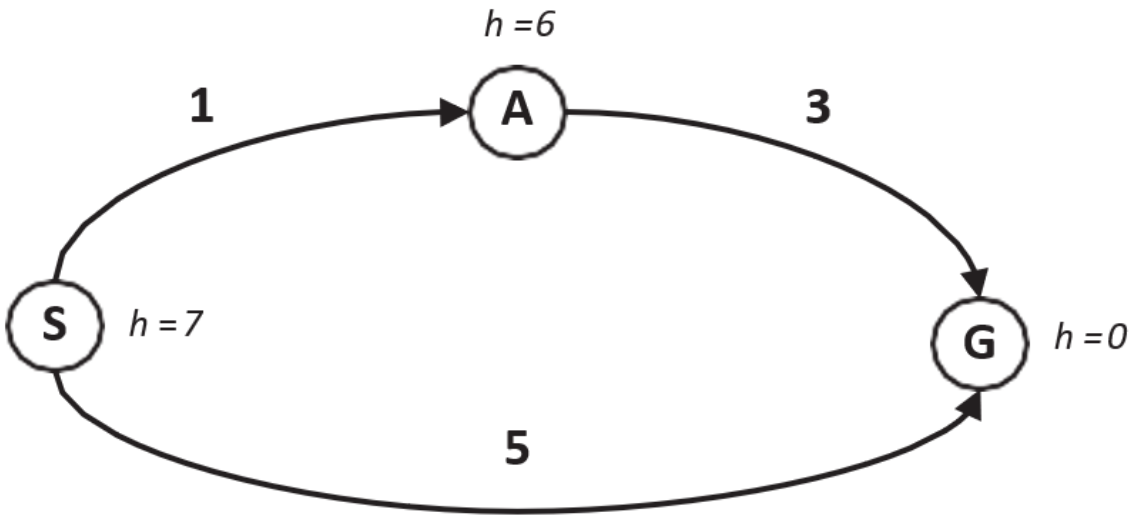
```
frontier = PriorityQueue()
frontier.put(start, 0)
came_from = dict()
cost_so_far = dict()
came_from[start] = None
cost_so_far[start] = 0

while not frontier.empty():
   current = frontier.get()

   if current == goal:
      break

   for next in graph.neighbors(current):
      new_cost = cost_so_far[current] + graph.cost(current, next)
      if next not in cost_so_far or new_cost < cost_so_far[next]:
         cost_so_far[next] = new_cost
         priority = new_cost + heuristic(goal, next)
         frontier.put(next, priority)
         came_from[next] = current
```

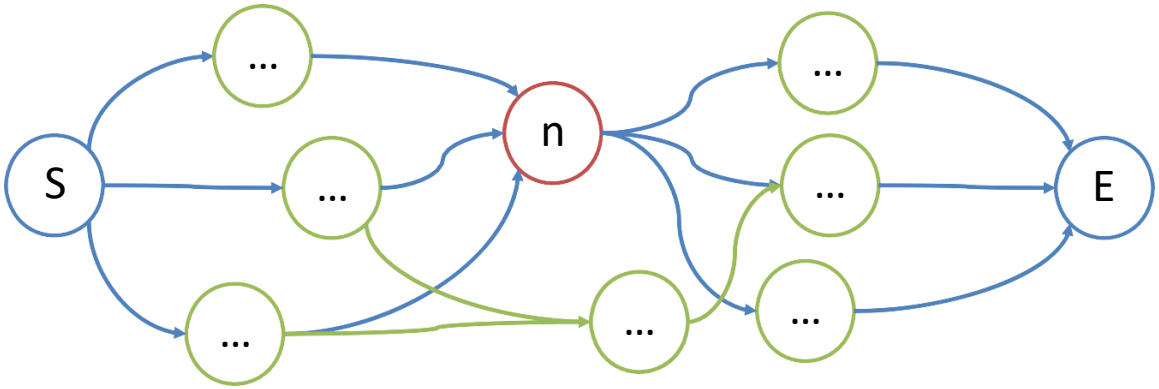
A搜索的解是最优的吗



- $f(A) = g(A) + h(A) = 1 + 6 = 7$
- $f(G) = g(G) + h(G) = 5 + 0 = 5$
- 为什么?
  - h(A)的估计太大了，甚至超出了实际cost，
  - 过大的启发值将淹没实际代价 g(n)，使得搜索脱离实际
  - 因此启发函数应该有上限

启发函数的上限问题

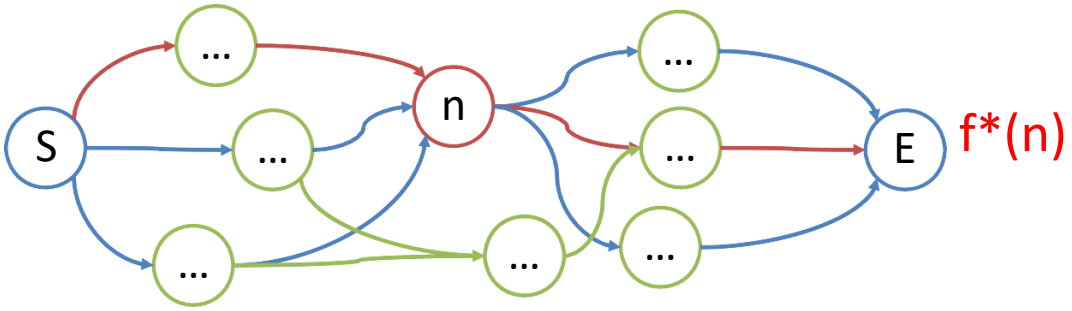
- A算法中，状态图上的每一步都有很多种可能的路径，我们希望能找到最优的路径。



- 最优路径满足什么条件？（假设n是最优路径上的点）

启发函数的上限问题

- g(n)是从 S 走到 n 的所有方式中，代价最小的路径，记为g*(n)
- h(n)是从 n 走到 E 的所有方式中，代价最小的路径，记为h*(n)
- 则最优路径为 f*(n)=g*(n)+h*(n)



启发函数的上限问题

- f*(n)无法直接计算，只能求近似，且需要能约束住算法估计值
  - 搜索过程中，g(n)是逐步优化得到的，因此 g*(n)≈g(n)是合理的
  - 有：g*(n) ≈ g(n)。

$$\begin{cases} f(n) \le f*(n) \text{(使得n点有被扩展的可能性)} \\ g*(n) \approx g(n) \end{cases}$$

  - 可以得出： h(n) ≦ h*(n)
  - 于是直观上理解，这就是启发函数的上限。
  - 如果启发函数大于这个上限，则搜索算法出现发散（跳过最优点），不能保证总能找到最优解。
  - h(n)=0 退化为一致代价搜索，则可找到最优解，h（n）趋于无穷则算法时效

1. **如果 $h(n)$ 高估了实际成本 $h^*(n)$**：这意味着算法可能会认为通过节点 n 的路径比实际上更糟，从而可能错过最短路径。因为如果 $h(n)$ 过大，$f(n)$ 也会相应变大，导致算法倾向于探索其他看似更有希望（即 f 值更小）的路径，而这些路径可能并不是最优的。
2. **如果 $h(n)$ 精确或低于实际成本 $h^*(n)$**：这样算法就不会错过任何潜在的最短路径，因为它总是偏向于探索那些估计总成本更低的路径。即使 $h(n)$ 低估了，最坏的结果就是算法会探索更多节点，这可能会使搜索过程更慢，但仍然可以保证找到最优路径。

在**A算法**中，如果代价函数f(n)=g(n)+h(n)始终满足 h(n)≦h*(n) 那么该算法就是**A*算法**。

h(n)的条件：可采纳性

I. 假设$c(n_1 \rightarrow n_2) \geq s > 0$. 每个状态转移（每条边）的成本是非负的，而且不能无穷地小

II. 假设$h^*(n)$是从节点n到目标节点的最优路径的成本 (当节点n到目标节点不连通时，$h^*(n) = \infty$)

■**当对于所有节点n，满足$h(n) \leq h^*(n)$，$h(n)$是可采纳的**

■所以，可采纳的启发式函数低估了当前节点到达目标节点的成本，使得实际成本最小的最优路径能够被选上；如果启发函数大于这个上限，则搜索算法出现 发散，不能保证总能找到最优解。

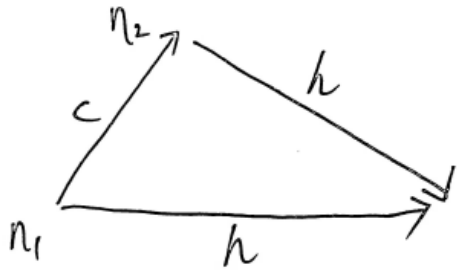
■因此，对于任何目标节点g，$h(g) = 0$

h(n)的条件：一致性 (单调性)

- 对于任意节点n_1和n_2，若

$$h(n_1) \leq c(n_1 \rightarrow n_2) + h(n_2)$$

则$h(n)$具有一致性/单调性



（想想，如果是大于号，代表的是不是过大估计了cost？）

- 注意到，满足一致性的启发式函数也一定满足可采纳性（证明如下）

Case 1: 从节点n没有路径到达目标节点，则可采纳性一定成立

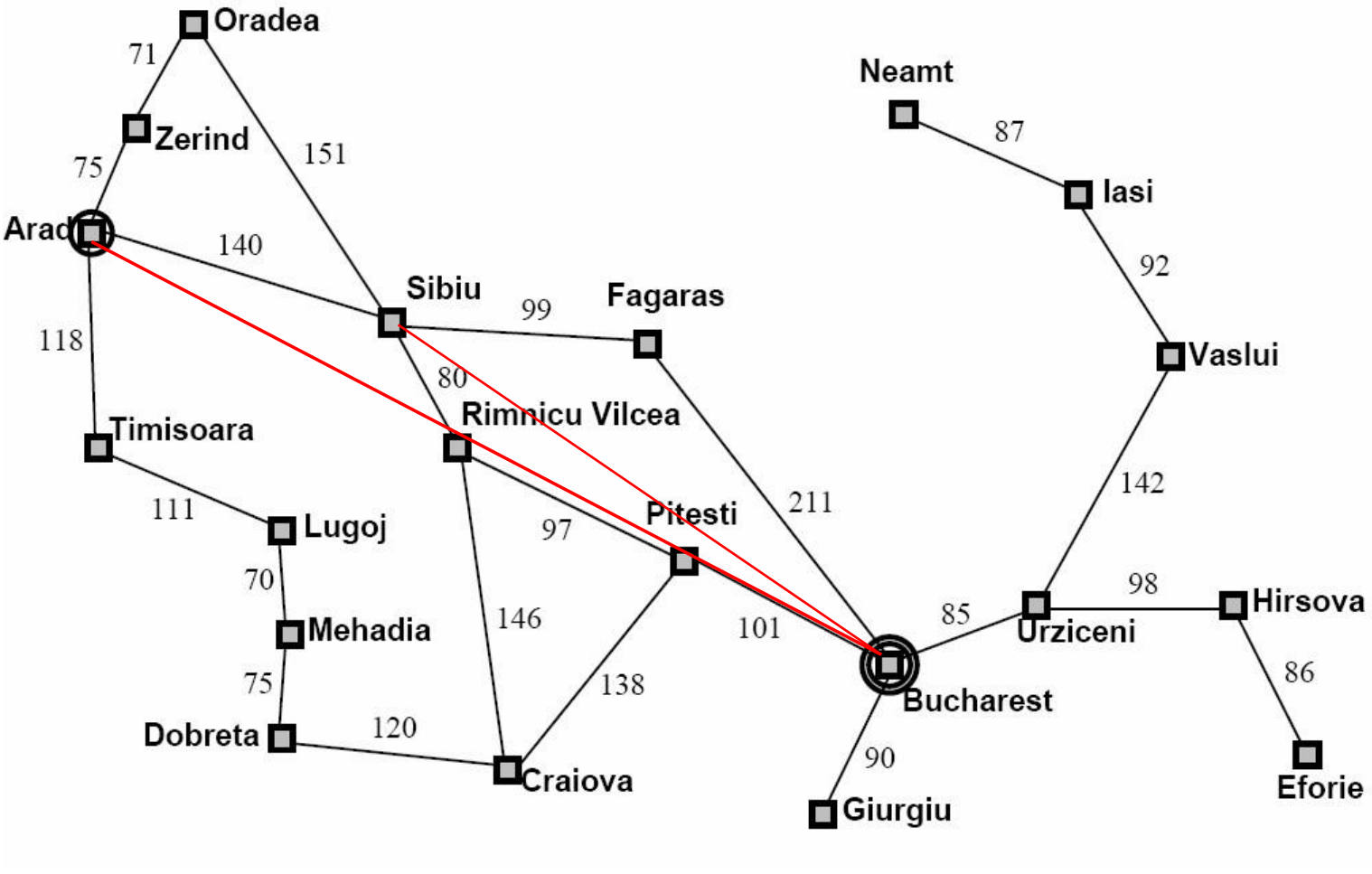
Case 2: 假设$n = n_1 \rightarrow n_2 \rightarrow \ldots \rightarrow n_k$是从节点$n$到目标节点的一条最优路径。可以使用数学归纳法证明对于所有的i，$h(n_i) \leq h^*(n_i)$.

Base:$h(n_k)=0$

Induction: $h(n_{i-1}) \leq c(n_{i-1} \rightarrow n_i) + h(n_i) \leq c(n_{i-1} \rightarrow n_i) + h^*(n_i) = h^*(n_{i-1})$

大部分的可采纳的启发式函数也满足一致性/单调性

示例：直线距离（是否满足一致性与可采纳性？）



时间和空间复杂度

- $h(n) = 0$时，对于任何n这个启发式函数都是单调的。 A*搜索会退化成一致代价搜索
- 因此一致代价的时间/空间复杂度的下界也适用于A*搜索。因此A*搜索仍可能是指数复杂度，除非我们能才找到好的h函数

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
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]

可采纳性意味着最优性

类似于一致代价搜索，我们也可以对A*算法最优性进行分析

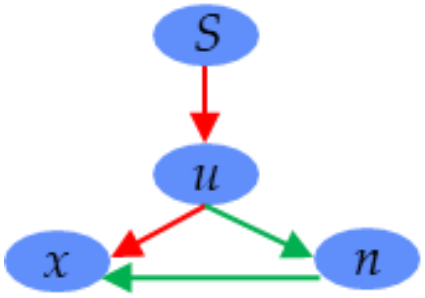
- 假设最优解的成本是C^*
- 最优解一定会在所有成本大于C^*的路径之前被扩展到(待证明)
- 成本$\leq C^*$的路径的数量是有限的
- 因此最终应该可以检测到最优解

可采纳性意味着最优性

最优解一定会在所有成本大于C^*的路径之前被扩展到

证明：

- 假设p^*是一个最优解的路径
- 假设p是一条满足$c(p) > c(p^*)$的路径，而且路径p在p^*之前被扩展（反证法）
- 那么扩展了到路径p时，肯定会有一个p^*上的节点n处在边界上
- 因为p在$p*$之前被扩展，因此p路径的最后的节点(假设为目标节点）x有
- $f(x) \leq f(n)$
- 因此
- $c(p) = g(x) + 0 = f(x) \leq f(n) = g(n) + h(n) \leq g(n) + h^*(n) = c(p^*)$
- 和$c(p) > c(p^*)$相矛盾



$p = S \rightarrow u \rightarrow x$

$p^* = S \rightarrow u \rightarrow n \rightarrow x$

示例：可采纳但不具备单调性的启发式函数

下面的启发式函数不是单调的，但是却是可采纳的，**why?**

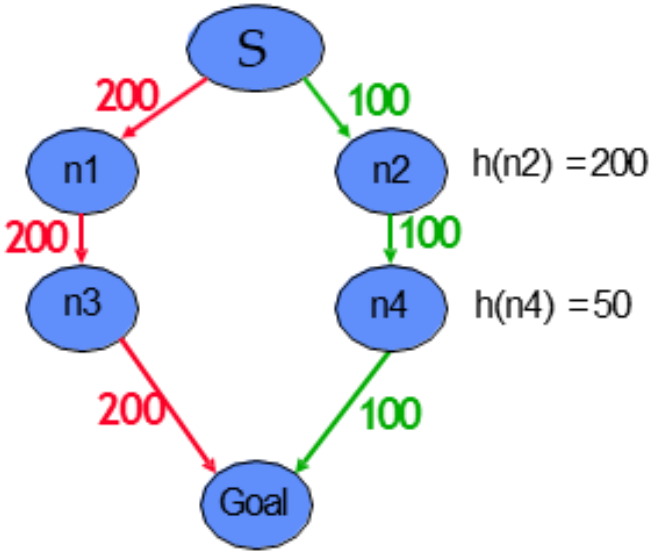
因为h(n2) > c(n2→n4) + h(n4)

→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**]}



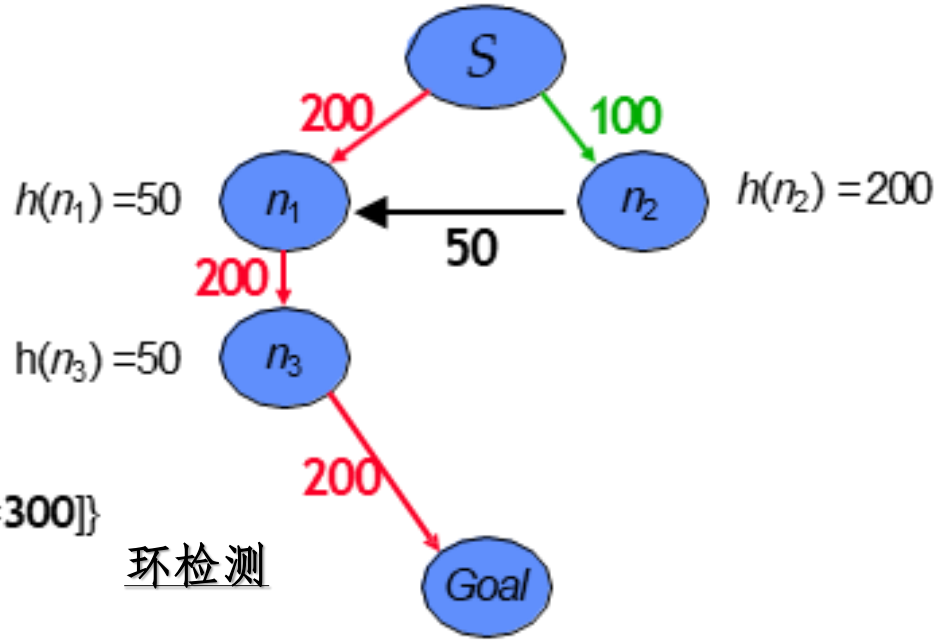
虽然确实可以找到最优路径，但是在搜索过程中错误地忽略了n2而去扩展n1 (Why？)

示例：可采纳但不具备单调性的启发式函数

下面的启发式函数不是单调的，但是却是可采纳的，因为h(n2) > c(n2→n1) + h(n1)

环检测的影响

→step cost=200
→step cost=100
→step cost=50



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

$\{S\} \rightarrow \{n_1\,[200+50=\mathbf{250}],\ n_2\,[200+100=\mathbf{300}]\}$
→
→
→

环检测

采用环检测：$S \rightarrow n_1 \rightarrow n_3 \rightarrow Goal$
$\searrow n_2$

最优路径：$S \rightarrow n_2 \rightarrow n_1 \rightarrow n_3 \rightarrow Goal$

环检测的影响

- 如果启发式函数只有可采纳性，则不一定能在使用了环检测之后仍保持最优性
- 为了解决这个问题：必须对于之前遍历过的节点，必须记录其扩展路径的成本。这样的话，若出现到达已遍历过节点但成本更低的路径，则需重新扩展而不能剪枝
- 启发式函数的一单调性可以保证我们在第一次遍历到一个节点时，就是沿着到这个节点的最优路径扩展的
- 因此，只要启发式函数具备单调性，就能在进行环检测之后仍然保持最优性

一致性相关结论

Proposition 1： 一条路径上的节点的 f 函数值应该是非递减的（注意如果只有可采纳性，此结论不成立）

证明：令$\ldots n1, n2, \ldots\ldots$为一路径

$$f(n1) = g(n1) + h(n1) \leq g(n1) + c(n1 \to n2) + h(n2)$$

$$f(n2) = g(n2) + h(n2) = g(n1) + c(n1 \to n2) + h(n2)$$

故$f(n1) \leq f(n2)$

一致性相关结论

Proposition 2： 如果节点$n2$在节点$n1$之后被扩展，则有

$$f(n1) \leq f(n2)$$

证明：

有以下两种情况：

- 当$n1$被扩展，$n2$还在边界上。由于$n2$在$n1$之后扩展，说明$f(n1) \leq f(n2)$
- 当$n1$被扩展，$n2$的祖先节点$n3$在边界上，则$f(n1) \leq f(n3)$。再根据Proposition 1，$f(n1) \leq f(n3) \leq f(n2)$

一致性相关结论

Proposition 3：在遍历节点n时，所有f值小于$f(n)$的节点都已经被遍历过了

证明：

- 假设存在路径 $p = n_1 \rightarrow n_2 \ldots \rightarrow n_k$还没有被遍历过，但

$$f(n_k) < f(n)$$

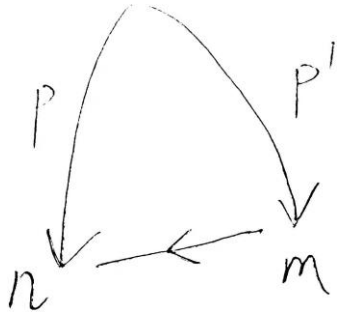
- 其中，n_k是路径 p上最后被遍历的节点
- 路径p上的节点n_{i+1}应该已经在n被探索时的边界上了，因此

$$f(n) \leq f(n_{i+1})$$

- 根据命题1，$f(n_{i+1}) \leq f(n_k)$
- 因此$f(n) \leq f(n_k)$，与假设矛盾

一致性相关结论

Proposition 4：A*搜索第一次扩展到某个状态，其已经找到到达该状态的最小成本路径



证明：

- 假设路径$p =\ldots\rightarrow\ n$是第一条被发现的到达n的路径
- 假设路径$p' =\ \ldots\rightarrow\ m\ \rightarrow\ n$是第二条被发现的到达$n$的路径
- 根据Proposition 2，$f(\mathrm{n})\ \mathrm{via}\ \mathrm{p} = g(p)\ +\ h(n)\ \leq f(m)$
- 根据Proposition 1，$f(m) \leq f(\mathrm{n})\ \mathrm{via}\ p' = g(p')\ +\ h(n)$
- 因此 $g(p) \leq\ g(p')$，即$c(p) \leq\ c(p')$

IDA* 迭代加深的A*算法

- A*搜索 和宽度优先搜索（BFS）或一致代价搜索（UCS）一样存在潜在的空间复杂度过大的问题
- IDA* - 迭代加深的A*搜索与迭代加深搜索一样用于解决空间复杂度的问题
- 就像迭代加深算法，但用于划定界限的不是深度，而是使用 *f* 值（*g*+*h*）
- 在每次迭代时，划定的界限是 *f* 值超过上次迭代的界限最少的节点的 *f* 值
- 可证明，当启发函数h为可采纳时， IDA* 是最优的： let C^* be the cost of the optimal solution, the cutoff value will be increased to C^*, and an optimal solution will be found

A* 搜索：总结

- 定义一个评价函数为$f\ (n)\ =\ g(n)\ +\ h(n)$
- 我们使用$f(n)$函数来对边界上的节点进行排序
- 可采纳性：$h(n)\ \leq\ h^{*}\ (n)$
- 一致性：对于任意节点$n1$和$n2$:
- $h(n1)\ \leq\ c(n1\ \rightarrow\ n2)\ +\ h(n2)$
- 启发式函数具有一致性说明其也具有可采纳性
- 启发式函数具有可采纳性说明其也具有最优性 (无环检测)
- A*搜索具有指数级的空间复杂度

一致性结论：总结

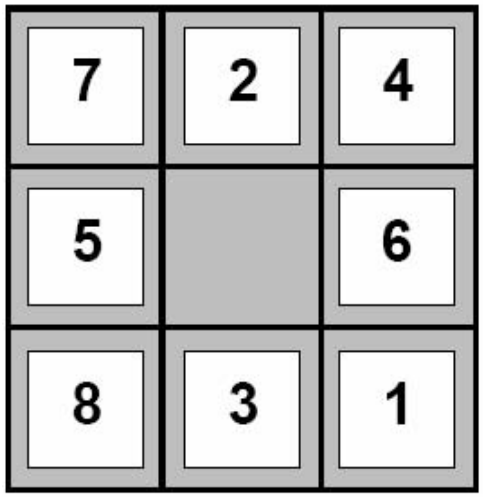
- 一条路径上的节点的 f 值应该是非递减的
- 如果$n2$节点在$n1$节点之后扩展，那么$f\,(n1)\ \leq\ f\,(n2)$
- A*搜索第一次扩展到某个状态，其已经找到到达该状态的最小成本的路径
- 只要启发式函数具备单调性（一致性），就能在进行环检测之后仍然保持最优性

构建启发式函数：松弛问题

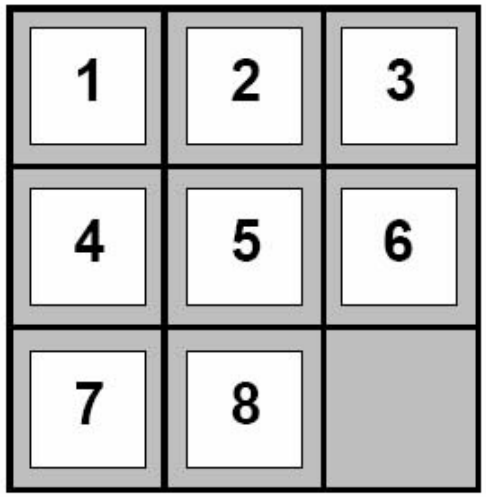
通过考虑一个比较简单的问题，并将 $h(n)$设置为简单问题中到达目标的成本

8数码问题：当满足下面条件时，可以把方块A移动到B位置

- A方块与B位置相邻（上/下/左/右相邻）
- B位置是空的



Start State



Goal State

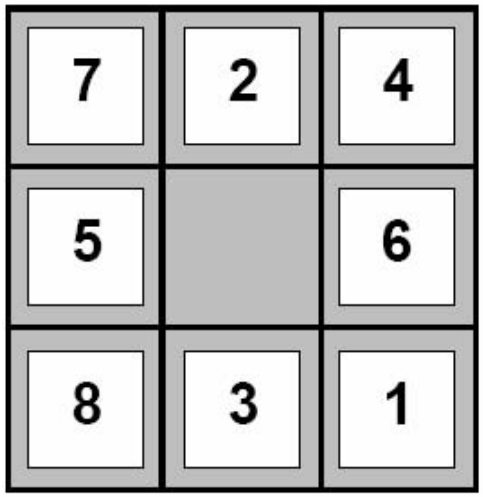
构建启发式函数：松弛问题

可以放松一些条件使得问题变简单

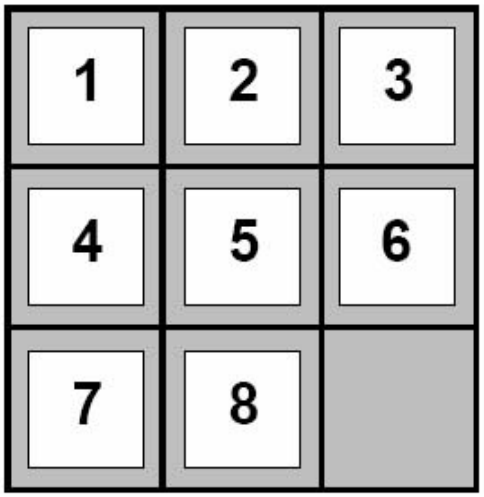
1. 只要方块A和B位置相邻就可以把A移动到B（不考虑B是否为空的条件）

2. 只要B位置为空的，就可以把A移动到B（忽略相邻的条件）

3. 任何情况下都可以把A移动到B（忽略两个条件）



Start State



Goal State

构建启发式函数：松弛问题

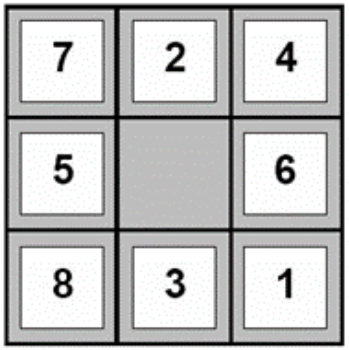
3. 任何情况下都可以把A移动到B（忽略两个条件）

#3 可以推导出**“不在目标位置方块数”(Misplaced)**的启发式函数

$$h(n) = \text{当前状态与目标状态位置不同的方块数}$$

可采纳性： 对于没有在目标位置上的方块，我们需要至少一次动作才能把其移动到目标位置，这个动作的成本大于等于1。所以$h(n) \leq h*(n)$

单调性： 任何动作都最多只能消除一个不在目标状态上的方块，因此对于任何8数字的状态，相邻状态位置不同的方块数最多差1，$h(n1) - h(n2) \leq 1 \leq c(n1 \rightarrow n2)$



Start State

Goal State

构建启发式函数：松弛问题

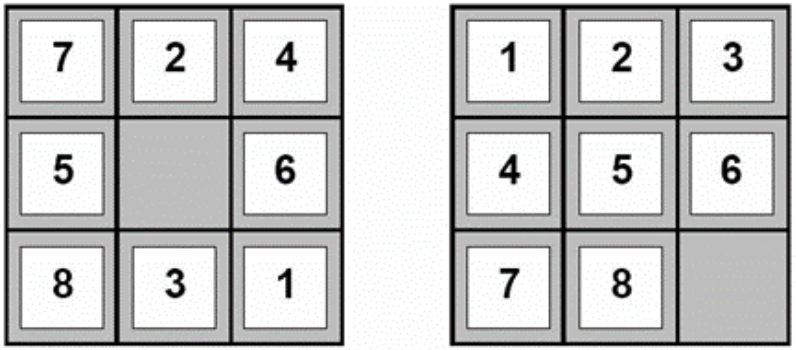
1. 只要方块A和B位置相邻就可以把A移动到B（不考虑B是否为空的）

#1 可以推导出**“曼哈顿距离”(Manhattan)**的启发式函数

$h(n)$ = 所有方块到达其目标位置的曼哈顿距离之和

可采纳性：对于每个不在目标位置的方块，都需要至少d个动作才能到达目标位置，其中d是该方块初始位置到目标位置的曼哈顿距离。不同的两个不在目标位置的方块，它们的这些动作是不同的，仍有$h(n) \leq h*(n)$

单调性：任何动作最多能使一个不在目标位置的方块的曼哈顿距离减少1，因此$h(n1) - h(n2) \leq 1 \leq c(n1 \rightarrow n2)$



构建启发式函数：松弛问题

Theorem. 在松弛问题中，到达某个节点的最优成本可作为原始问题中到达该节点的可采纳的启发式函数值

证明：

- 若P是一个初始问题，设P_j是问题P的松弛问题
- 那么 $Sol(P) \subseteq Sol(P_j)$,$Sol(P)$表示问题$P$的解集
- 于是$mincost\left(Sol(P_j)\right) \leq mincost(Sol(P))$
- 因此 $h(n) \leq h^*(n)$

比较不同启发式函数

Definition. 假如启发式函数h1和h2都是可采纳的，并且对于除了目标节点之外的其他节点，都有$h1(n) \leq h2(n)$，我们称h2函数**支配**了h1函数（或者h2函数比h1含有更多信息，更逼近于h*）

Theorem. 假如h2函数**支配**了h1函数，那么在使用A*算法时，使用h2函数扩展的节点，使用h1函数也会扩展到。

Depth	IDS	A*(Misplaced) **h1**	A*(Manhattan) **h2**
10	47,127	93	39
14	3,473,941	539	113
24	---	39,135	1,641

- 启发函数如果与问题求解越接近越好

使用带环检测的A*算法解决8数码问题

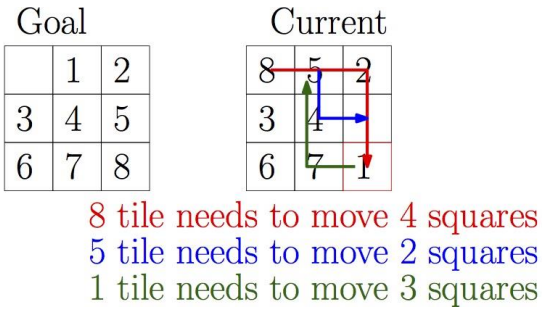
采用**Manhattan启发式函数**$h(n)$，用**带环检测的A*搜索**初始状态和目标状态如下图所示的8数码问题，画出搜索图，图中标明所有节点的*f*值

初始：

2	8	3
1	6	4
7		5

目标：

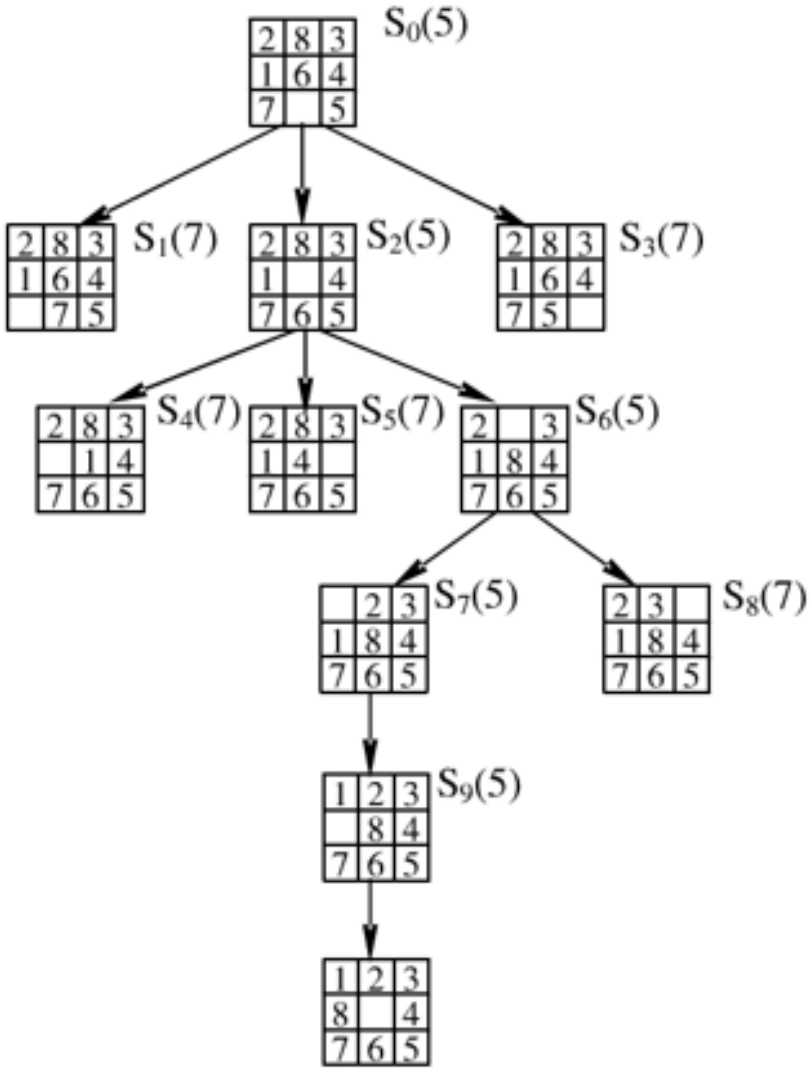
1	2	3
8		4
7	6	5



$h(n)$ = 所有方块到达其目标位置的曼哈顿距离之和

The Current state has a manhattan distannce of 9 to the goal state in the above example

使用带环检测的A*算法解决8数码问题



循环	OPEN	CLOSED
初始化	S_0	
1	$S_2\ S_1\ S_3$	S_0
2	$S_6\ S_1\ S_3\ S_4\ S_5$	$S_0\ S_2$
3	$S_7 S_1\ S_3\ S_4\ S_5\ S_8$	$S_0\ S_2\ S_6$

搜索树如左图（右上角的数字是其估价函数值）

初始：

2	8	3
1	6	4
7		5

目标：

1	2	3
8		4
7	6	5

积木世界规划

现有积木若干，积木可以放在桌子上，也可以放在另一块积木上面。有两种操作：

1. $move(x, y)$：把积木x放到积木y上面。前提是积木x和y上面都没有其他积木。
2. $moveToTable(x)$：把积木x放到桌子上，前提是积木x上面无其他积木，且积木x不在桌子上。

设计本问题的一个可采纳的启发式函数，然后用A*搜索初始状态和目标状态如下图所示的规划问题：

积木世界规划

- 积木处于其目标位置：以该积木为顶的塔出现在目标状态中
- 启发式函数：令h(n)为状态n中不在目标状态的积木数
- 可采纳性：对于每个不在目标位置的积木，需要至少1步动作来使得其到达目标位置。每块不在目标位置的积木要移动到目标位置的动作都不相同（即不存在一个动作使得两块积木同时到达目标位置的情况）
- 单调性：任何动作最多都只能消除一个不在目标位置的积木

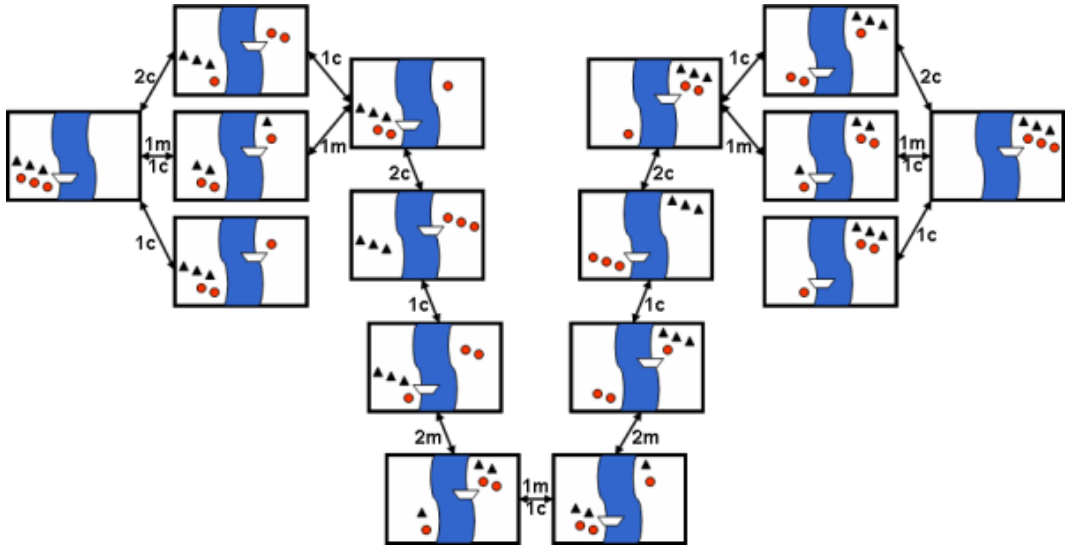
积木世界规划

- 是否可以设计一个更好的具有可采纳性的启发式函数?
  - （考虑下面的策略）
- 当积木x已经处于其目标位置，我们说x是一个good tower
- 如果当前状态下采取某一个动作可以创建一个good tower，则进行该动作；
- 否则，把一个积木移到桌上，但要确保移动的这个积木不是good tower

传教士和野蛮人的问题

- 有N 个传教士和N个野蛮人在河的左岸
- 有一艘可以载K个人的小船
- 寻求一种可以把所有人运到河的右岸的方法
- 并且要求无论何时何地 (在河的任意岸或在小船上),

  传教士的人数 ≥ 野蛮人的人数 或 传教士的人数 = 0



形式化传教士和野蛮人的问题

- 状态 (M, C, B) 表示：M － 左岸的传教士的人数，C －左岸的野蛮人的人数，B＝1表示小船在河的左岸
- 动作 (m, c) 表示：m － 小船上的传教士的人数，c －小船上的野蛮人的人数
- 前提条件：传教士的人数和野蛮人的人数满足题目中的约束
- 动作的效果

$$(\mathrm{M},\ \mathrm{C},\ 1) \rightarrow (m,c) \rightarrow (M-m, C-c, 0)$$

$$(\mathrm{M},\ \mathrm{C},\ 0) \rightarrow (m,c) \rightarrow (M+m, C+c, 1)$$

对于K ≤ 3时的启发式函数

$h1(n) = M + C$的启发式函数是可采纳的吗?

不是，考虑状态$(1,1,1)$时，

$h1(n) = 2$，但$h^*(n) = 1 < 2$

假设$h(n) = M + C - 2B$

单调性:

$$(\mathrm{M},\ \mathrm{C},\ 1) \to (m,c) \to (M-m, C-c, 0)$$

$$h(n_1) - h(n_2) = m + c - 2 \le K - 2 \le 1$$

$$(\mathrm{M},\ \mathrm{C},\ 0) \to (m,c) \to (M+m, C+c, 1)$$

$$h(n_1) - h(n_2) = 2 - (m+c) \le 1, since\ m + c \ge 1$$

直接证明可采纳性

当 B＝1,

在最好的情形下,我们在最后一步把3个人送到河的右岸

在此之前，我们可以把三个人送到右岸，再由一个人把船摆渡到左岸。因此，每次来回都只能渡2个人过河，

因此，我们需要$\geq 2\left|\frac{M+C-3}{2}\right|+1 \geq M+C-2$个动作

当 B＝0,

我们需要一个人将船摆渡到左岸，这样就形成了B＝1的情形。

现在我们需要把$\mathrm{M}+\mathrm{C}+1$的人摆渡到右岸

因此，我们总共需要$\geq \mathrm{M}+\mathrm{C}+1-2+1=\mathrm{M}+\mathrm{C}$个动作

直接证明可采纳性

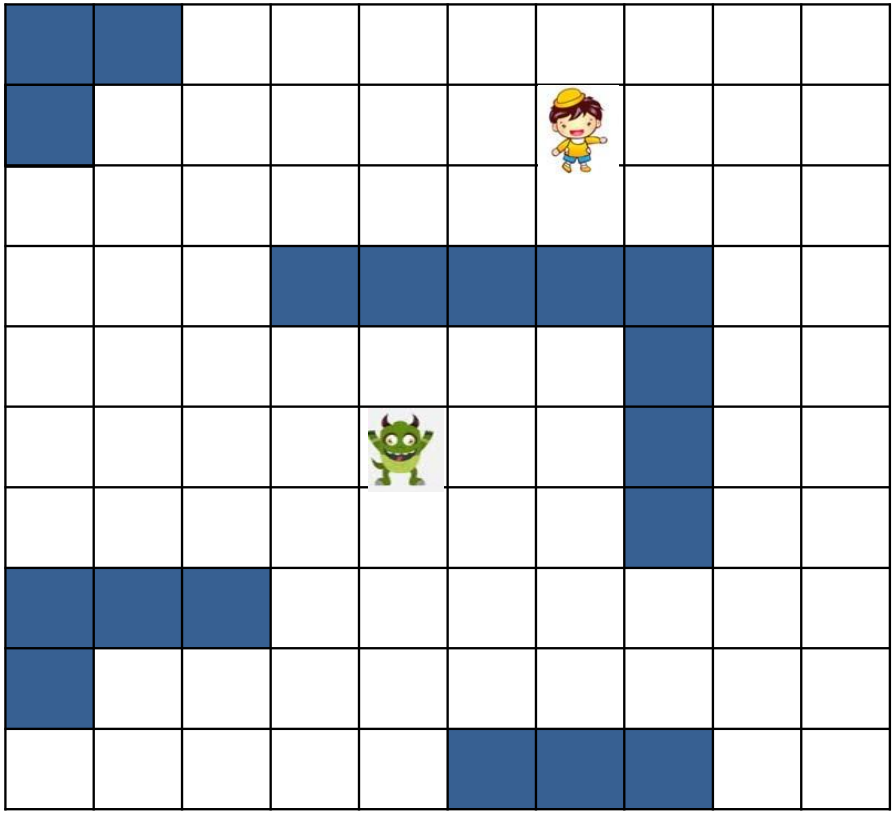
- 两者结合，得到：

$$h(n) = \begin{cases} M + C - 2, B = 1 \\ M + C, \quad B = 0 \end{cases}$$

- 综合即 $h(n) = M + C - 2B$ 此时满足A*条件

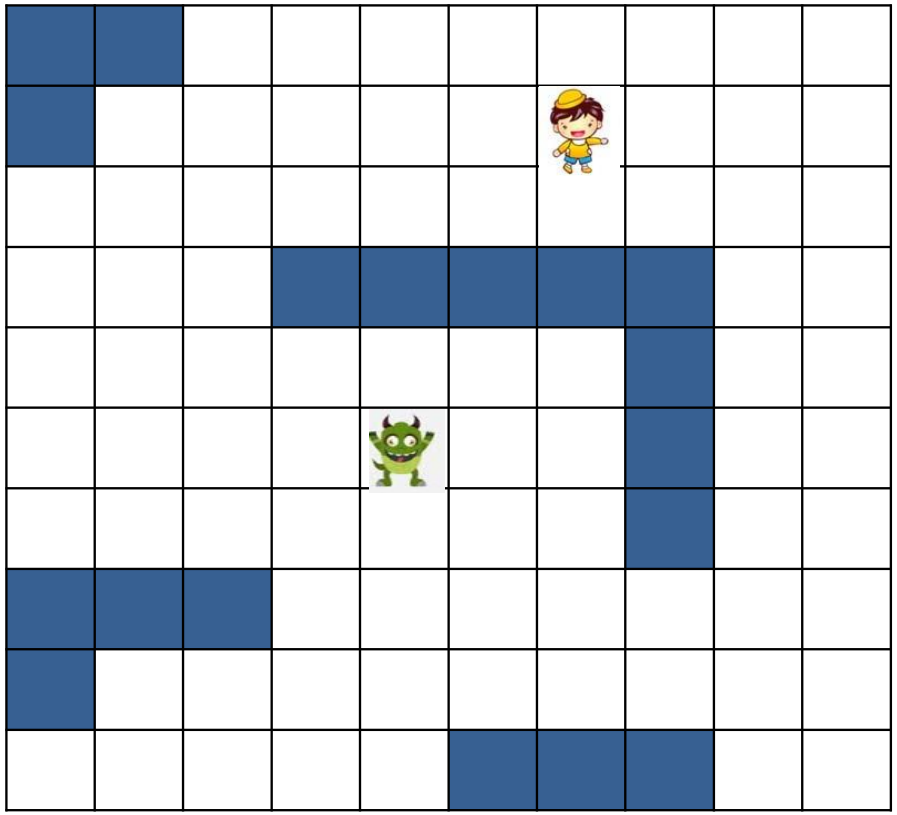
游戏中的怪物追逐

- 游戏中，怪物试图靠近人发起攻击，应该沿着什么路径过来遇到障碍物该怎么办？
  - 怪物可以横向、纵向、斜向运动
  - 假定小孩不动
  - 如何找到最快接近的路线？



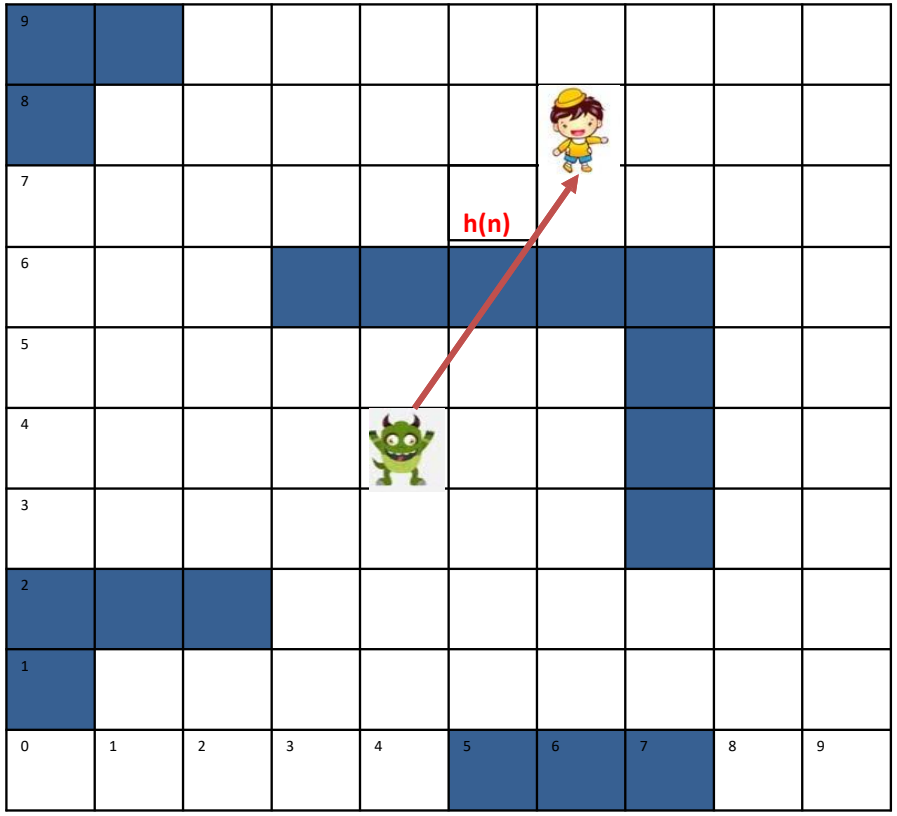
游戏中的怪物追逐

- Step1： 状态空间和状态转移
  - 直接用怪物位置作为状态
  - 直接用坐标点作为转移动作



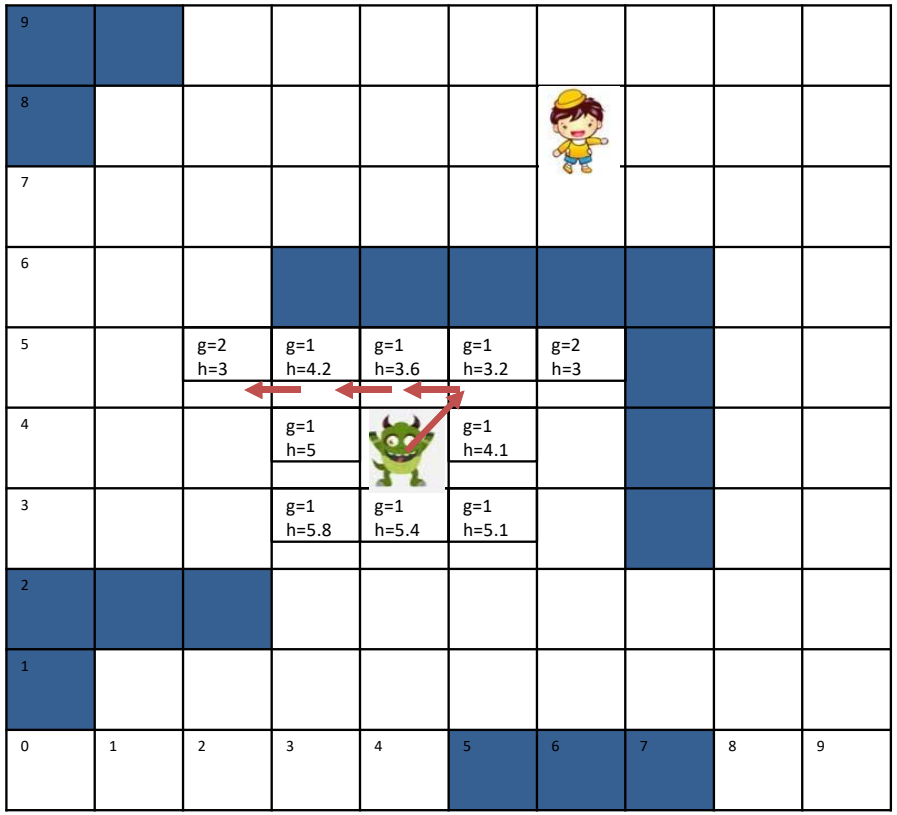
游戏中的怪物追逐

- Step2： 设计A*启发函数
  - 优化目标为步骤最短。
  - 两点之间直线最短，因此可以设计启发函数为：怪物当前坐标与小孩目标坐标的直线距离。
  - 如图中(4,4)到(8,6)距离为4.47
  - 显然，此时满足 h(n)≤h*(n)



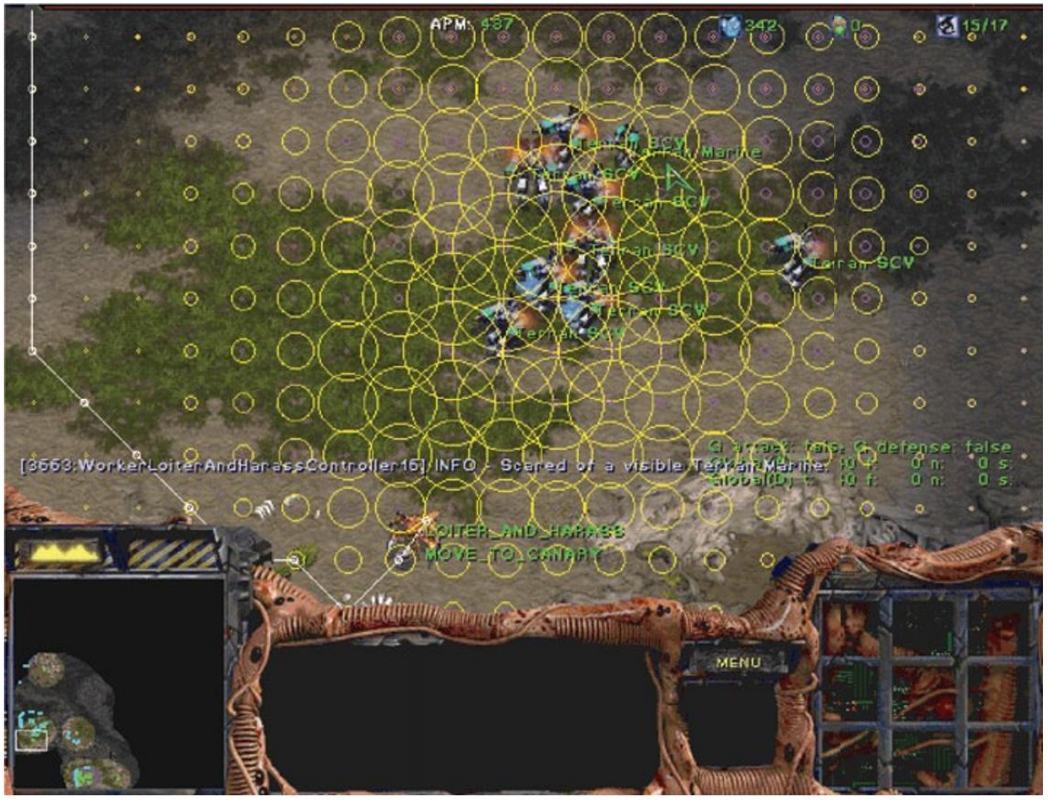
游戏中的怪物追逐

- Step3： 计算搜索过程
  - 在0时刻，向8个方向行动1步，g = 1
  - 计算每种可能下的h值
  - 选择最优情况前进。
  - 最终路线如何？



A*搜索应用

- 游戏AI
- 导航路径
- 资源分配
- 机器人运动
- 语言分析



A*搜索应用

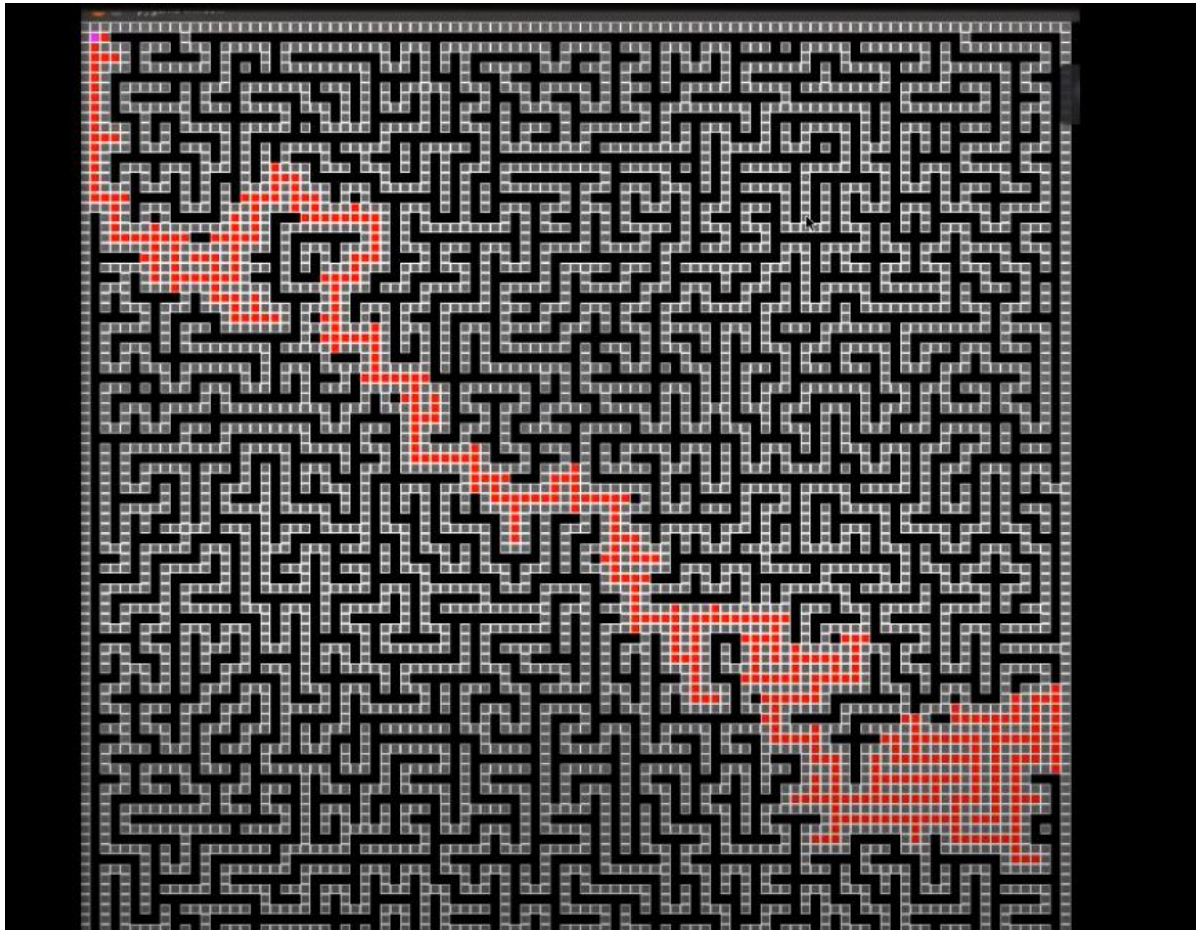
https://www.youtube.com/watch?v=2XjzjAfGWzY

A*搜索应用

Maze solver using A* pathfinder algorithm

https://www.youtube.com/watch?v=J-ilgA_XNI0

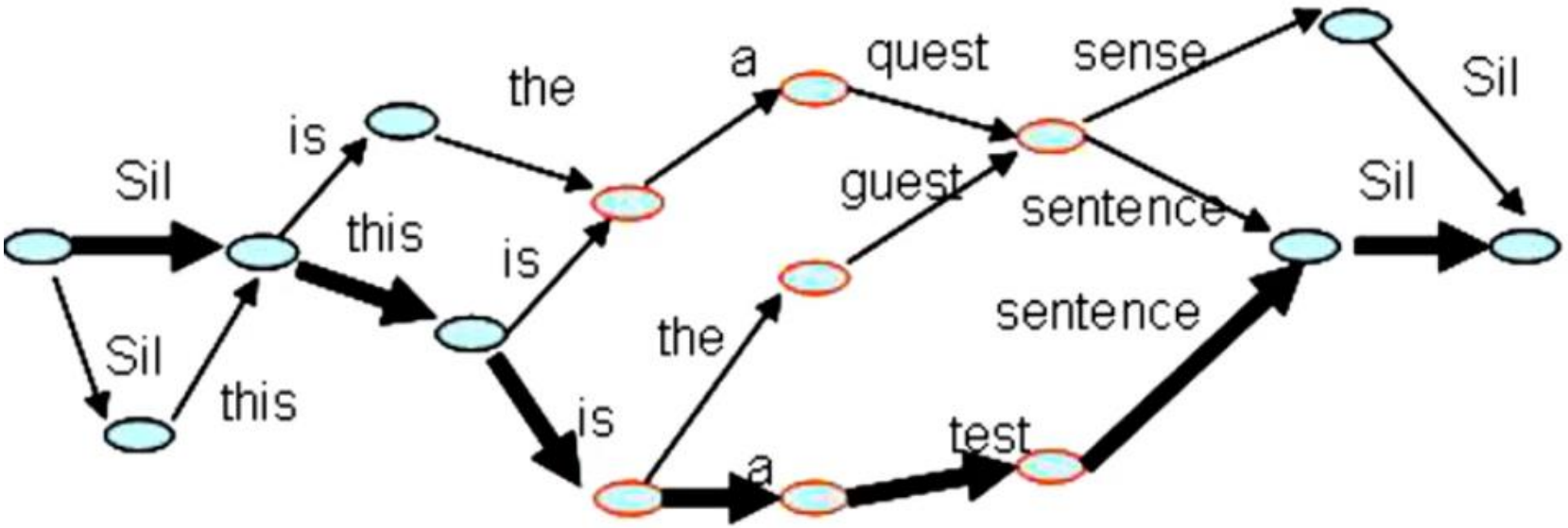
https://github.com/MateusZitelli/PAstar/blob/master/PAstar.py



A*搜索应用

语音词图中的启发式搜索

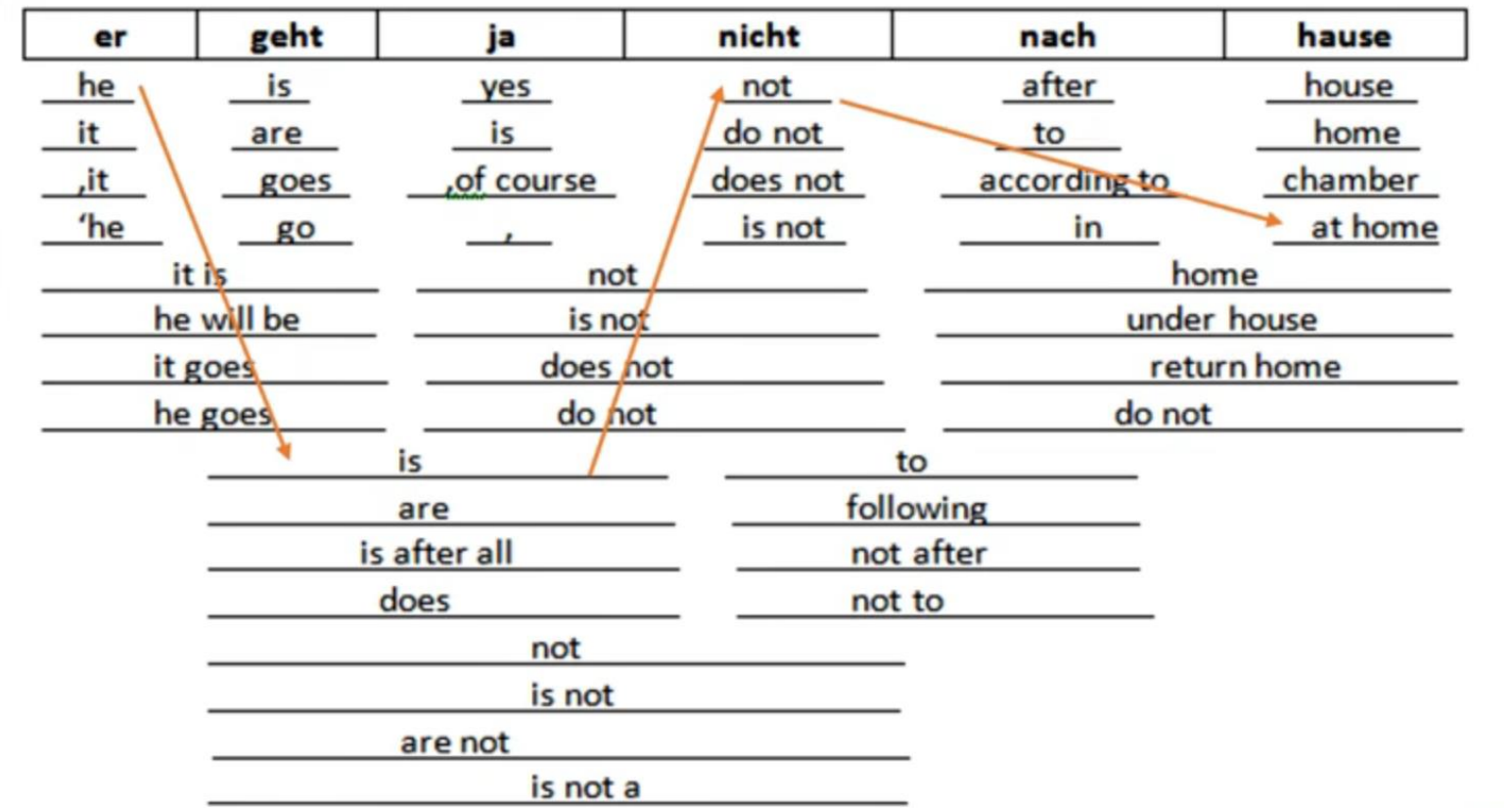
在统计语音识别过程中，计算机将用户的每个发音转化成若干可能的词，构成如下的词图（比如sense、sentence）。语音识别的任务就是要在这样的词图中寻找最有可能组成合理句子的组合。



A*搜索应用

机器翻译中的启发式搜索

源语言的若干可能的翻译形式需要通过拼接，形成最终的翻译译文。每一个外文词都有很多种翻译方法，所有翻译词形成一个状态空间，最佳译文就是在状态空间中找到一个最佳的搜索路线。



Thanks

Thanks the support from Prof. Rao, Prof. Liu, Prof. Sheila McIlraith and Prof. Andrew Moore, for the refer to their slides.