



回顾:树搜索

```
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
```

- 搜索策略就是节点扩展顺序的**选择**

有信息搜索

- **无信息搜索** 又名盲目搜索:

- 在搜索时, 只有问题定义信息可用。
- 盲目搜索策略仅利用了问题定义中的信息。

- **有信息搜索:**

- 在搜索时, 当有**策略**可以确定一个非目标状态比另一种更好的搜索, 称为有信息的搜索。



5.1 最佳优先搜索

- **思想:** 使用一个评估函数 $f(n)$ 给每个节点估计他们的希望值。 优先扩展最有希望的未扩展节点。
- **实现:** **fringe**表中节点根据评估值从大到小排序
- **最佳优先**搜索策略有:
 - 贪婪最佳优先搜索
 - A^* 搜索





贪婪最佳优先搜索

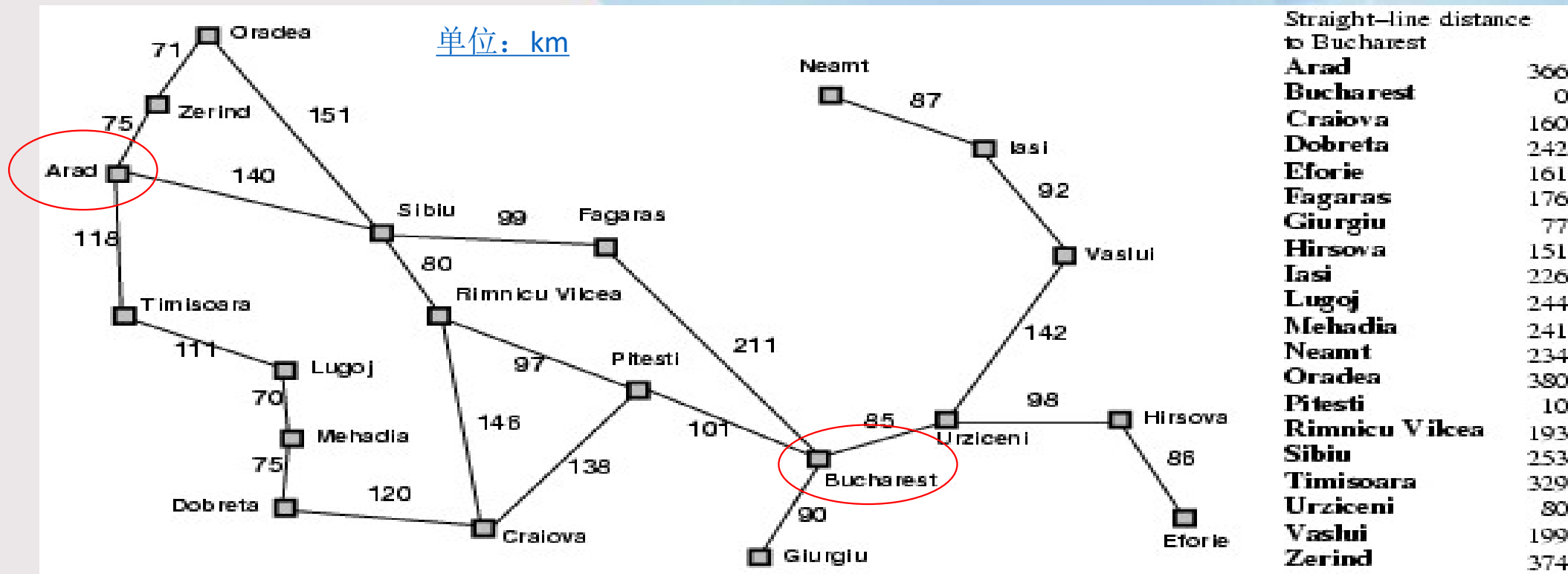
- 评估函数: $f(n) = h(n)$ (**h**euristic, 启发函数)
= **估计**从节点n到目标的代价

例如: $h_{SLD}(n)$ = 从节点 n 到 Bucharest 的
直线距离

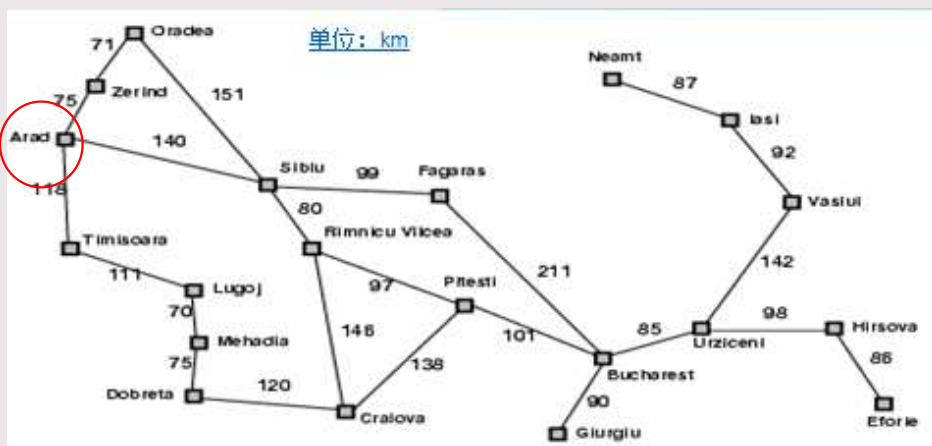
- **贪婪最佳优先搜索** 优先扩展看上去更**接近目标**的节点（启发式函数评估出来的）。

例：罗马尼亚问题

单位：km

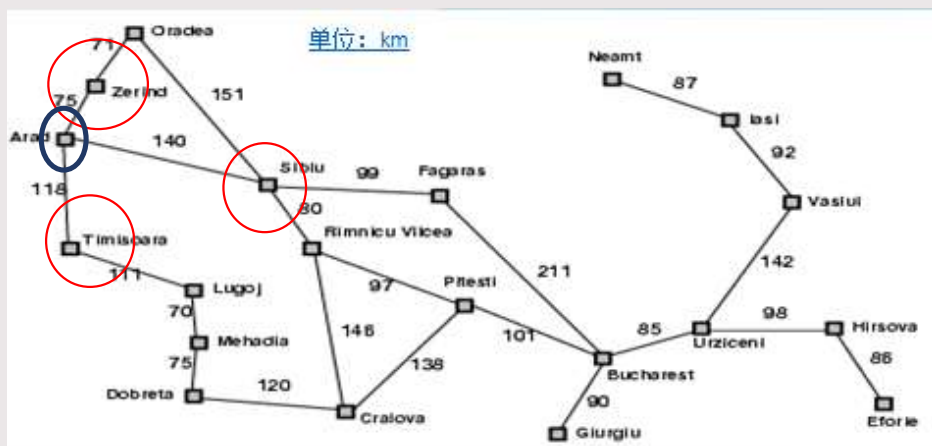
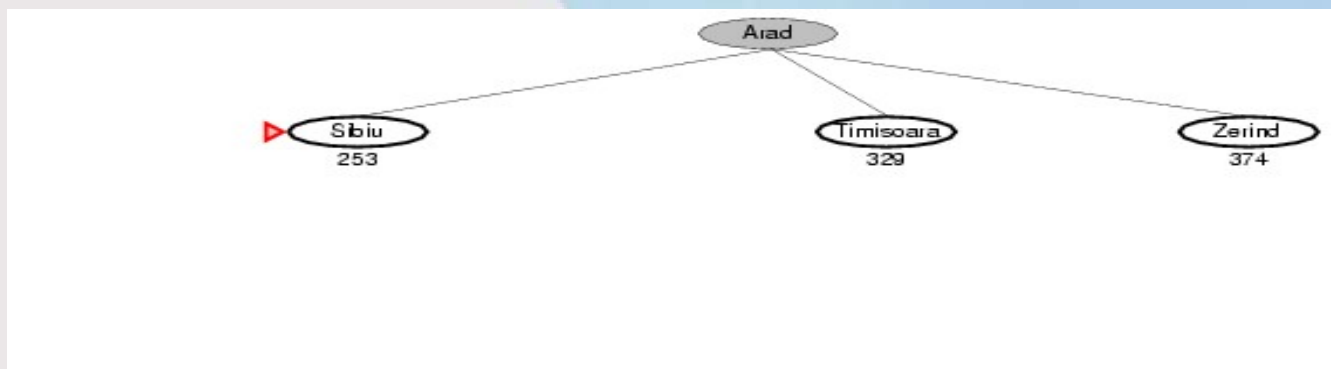


贪婪最佳优先搜索示例



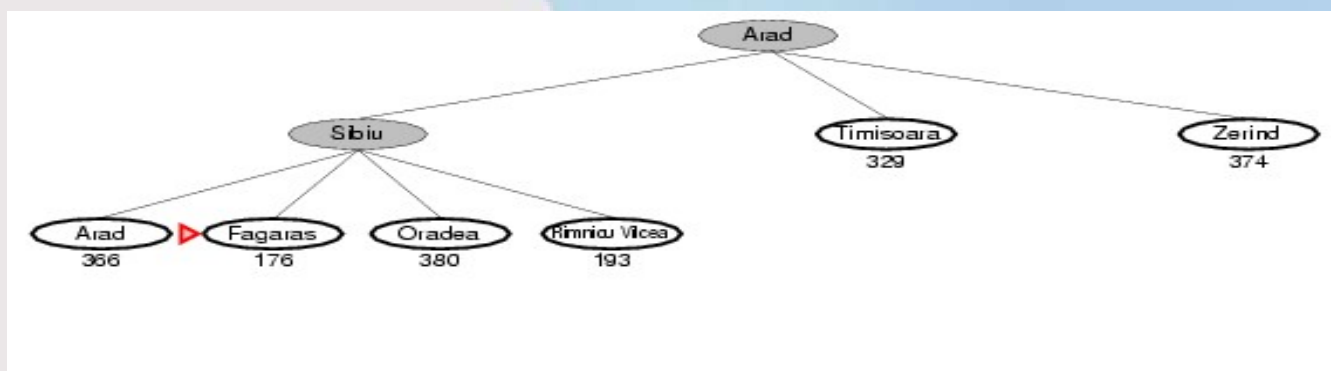
Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

贪婪最佳优先搜索示例

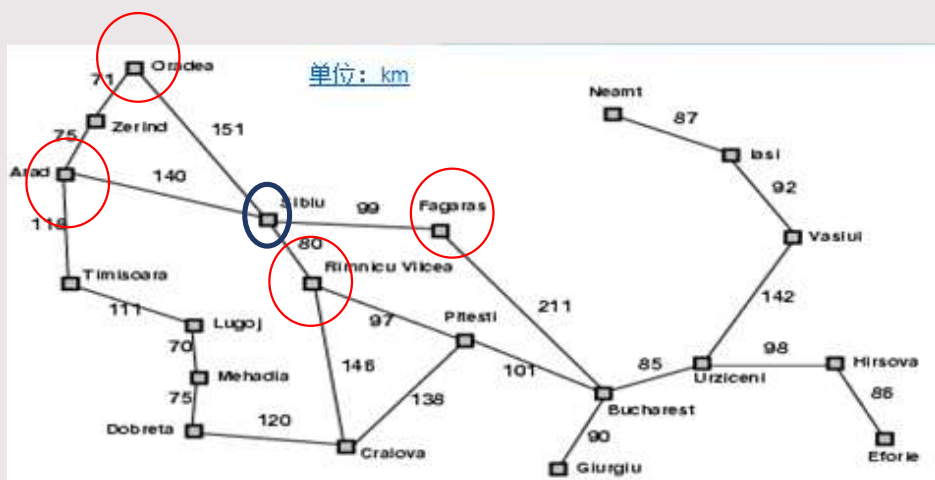


Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

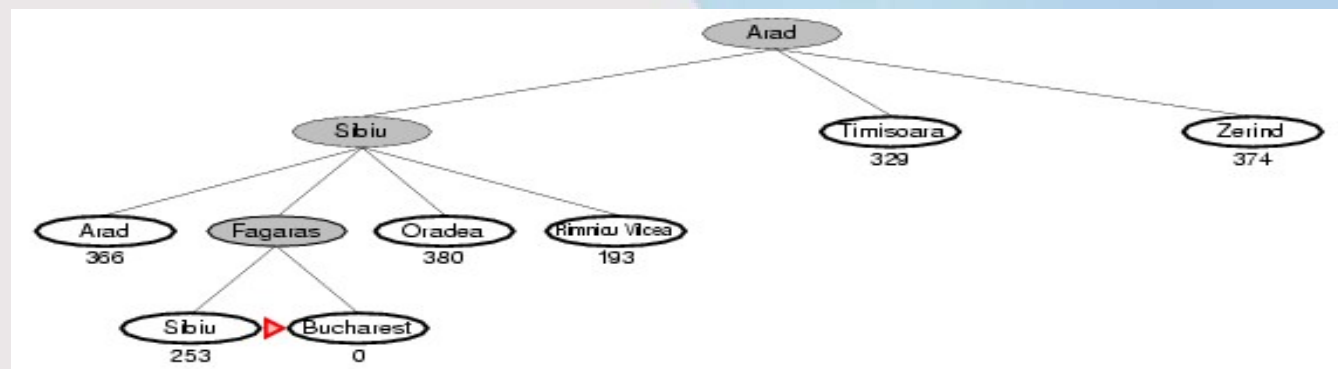
贪婪最佳优先搜索示例



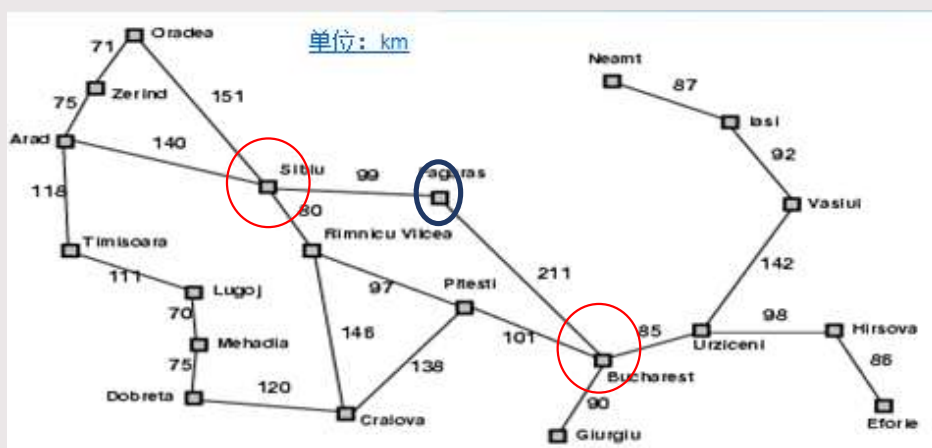
Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374



贪婪最佳优先搜索示例



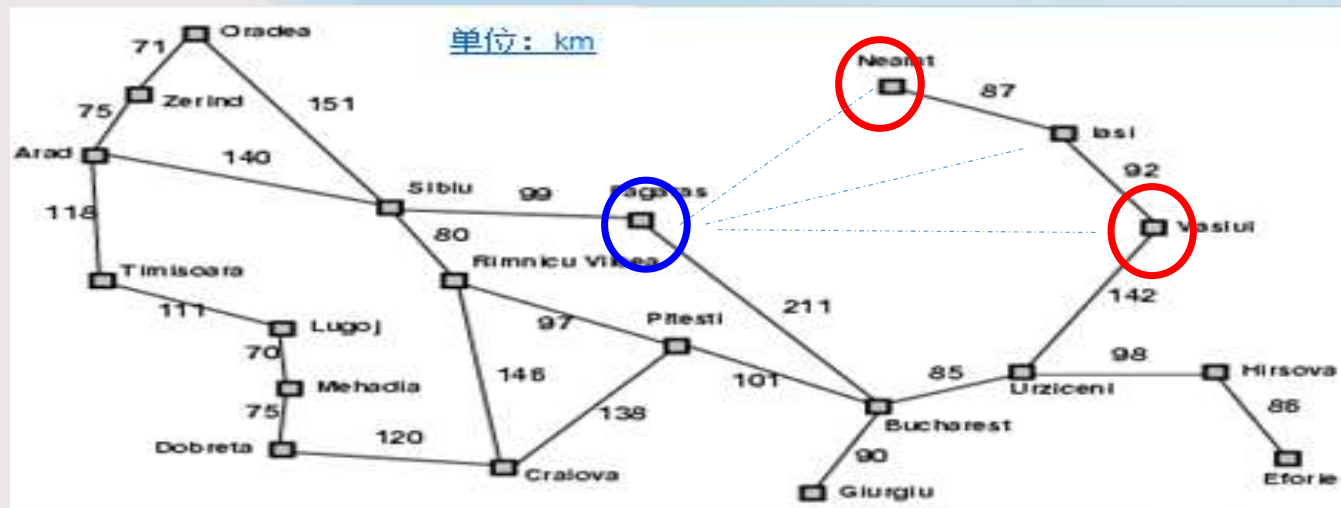
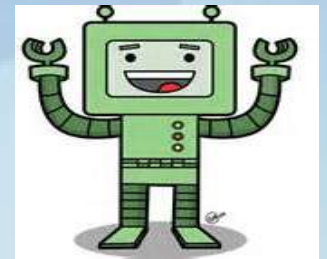
Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374



路径代价: $140+99+211=450$

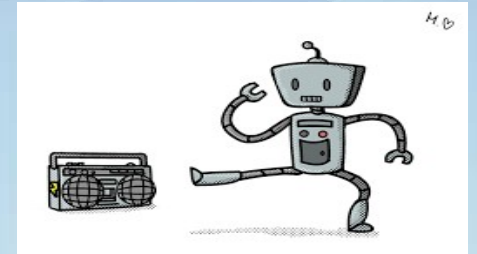
贪婪最佳优先搜索的属性

- 完备性? No – 可能陷于死循环当中,
比如, Iasi → Neamt → Iasi → Neamt →
- 时间? $O(b^m)$, 但一个好的启发式函数能带来巨大改善
- 空间? $O(b^m)$
- 最优性? No




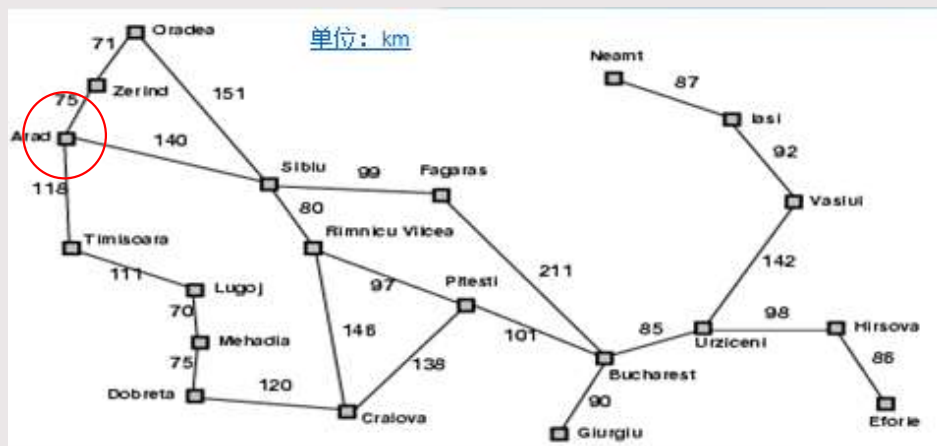
5.2 A* 搜索

- **思想:** 避免扩展代价已经很高的节点。
- 评估函数 $f(n) = g(n) + h(n)$
 - $g(n)$ = 到达节点 n 已经发生的实际代价
 - $h(n)$ = 从节点 n 到目标的代价估计值
 - $f(n)$ = 评估函数, 估计从初始节点出发, 经过节点 n , 到目标的路径代价的估计



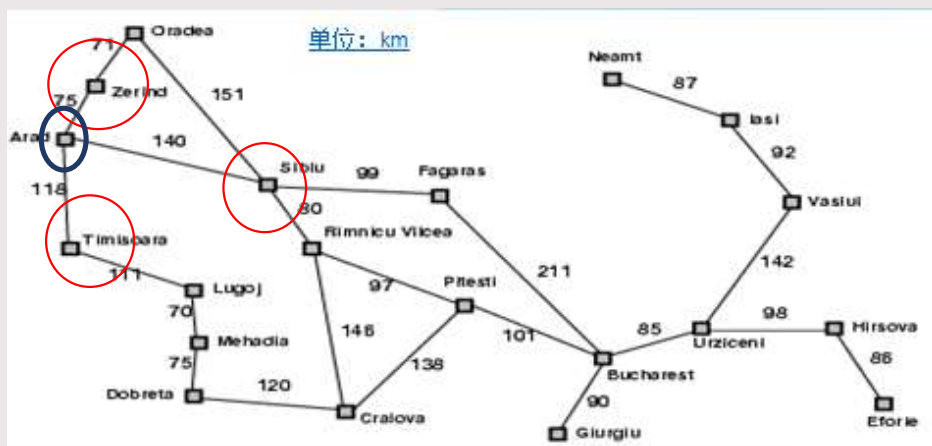
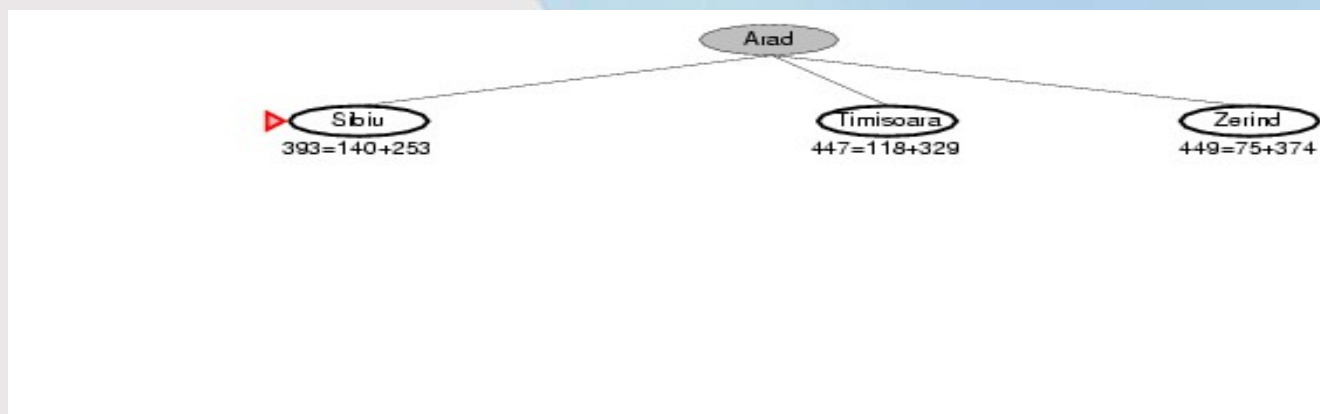
A* 搜索示例


 Arad
 $366 = 0 + 366$



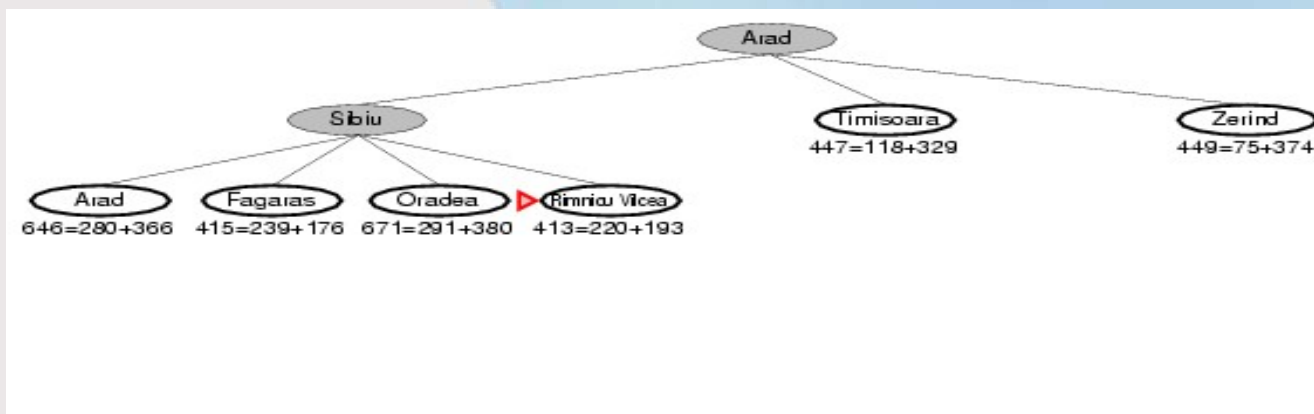
Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

A*搜索示例

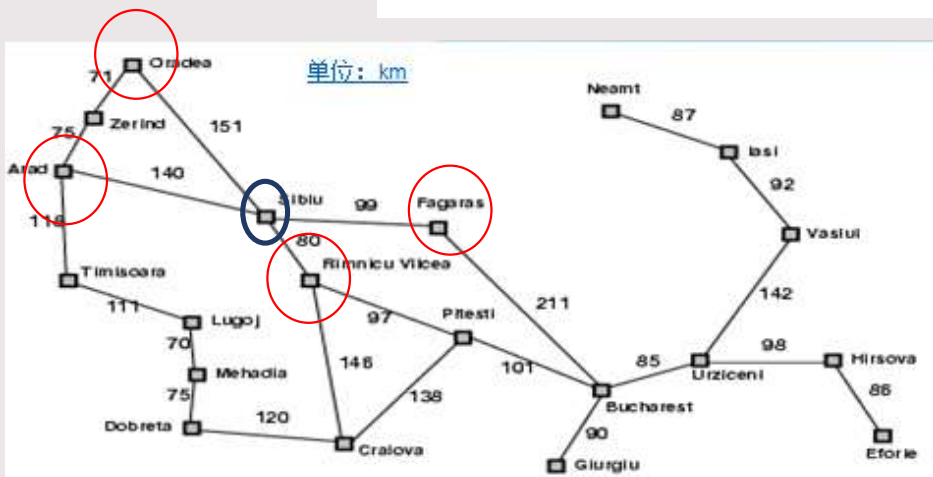


Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

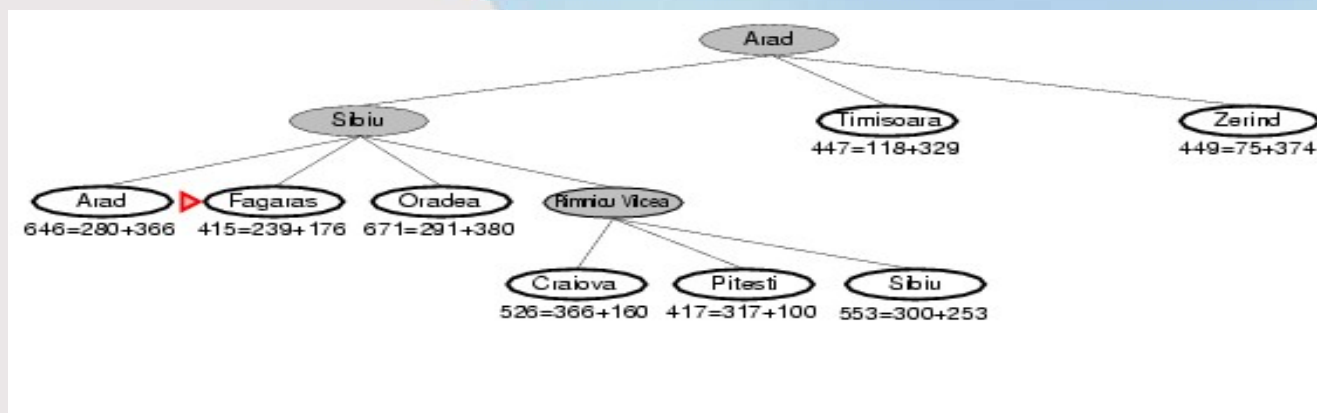
A*搜索示例



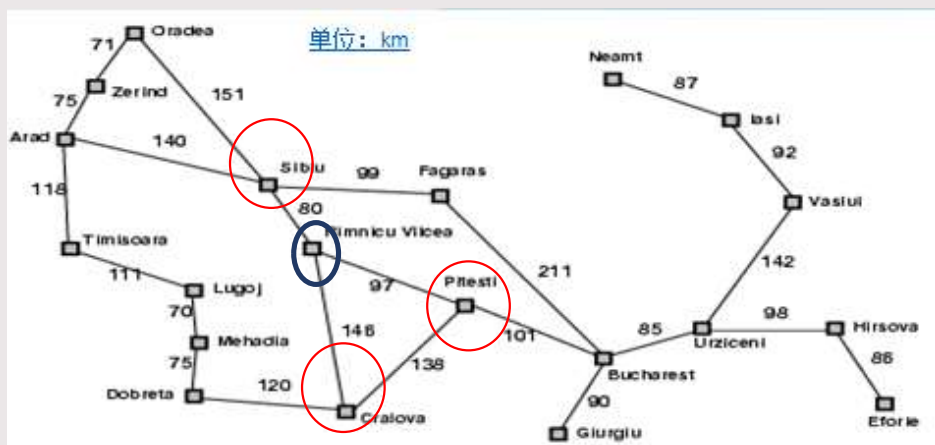
Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374



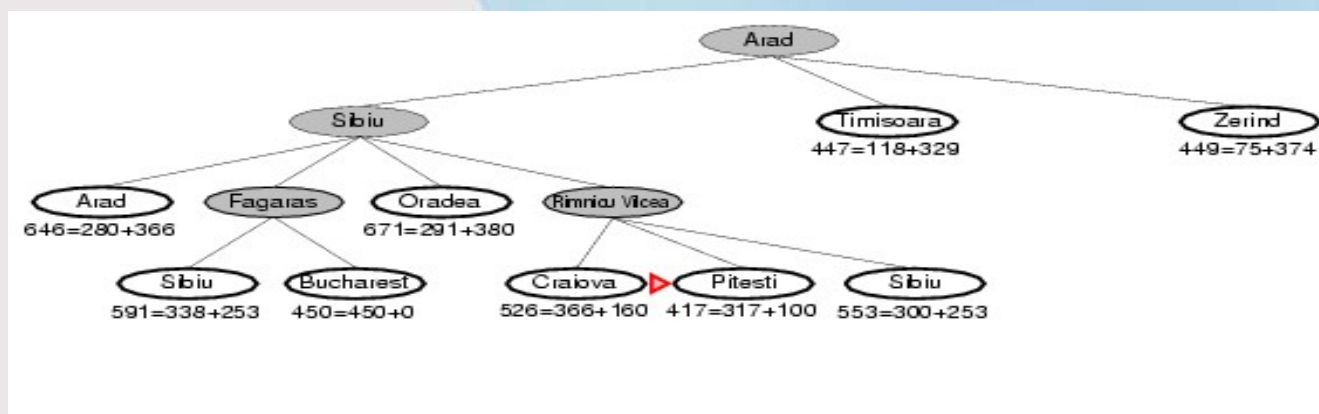
A*搜索示例



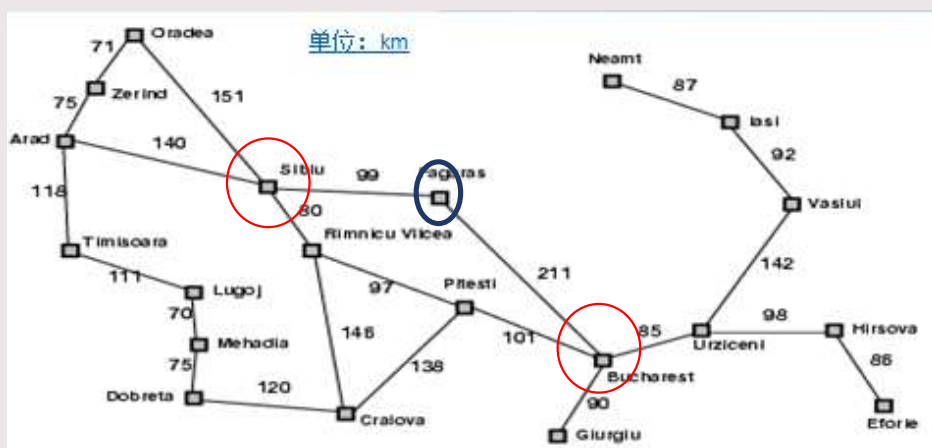
Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374



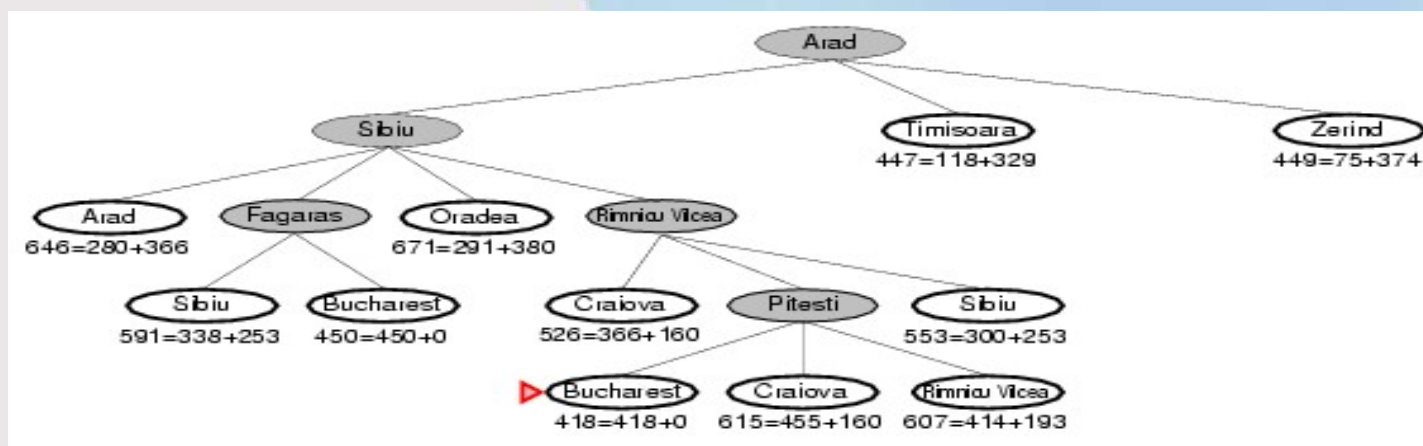
A*搜索示例



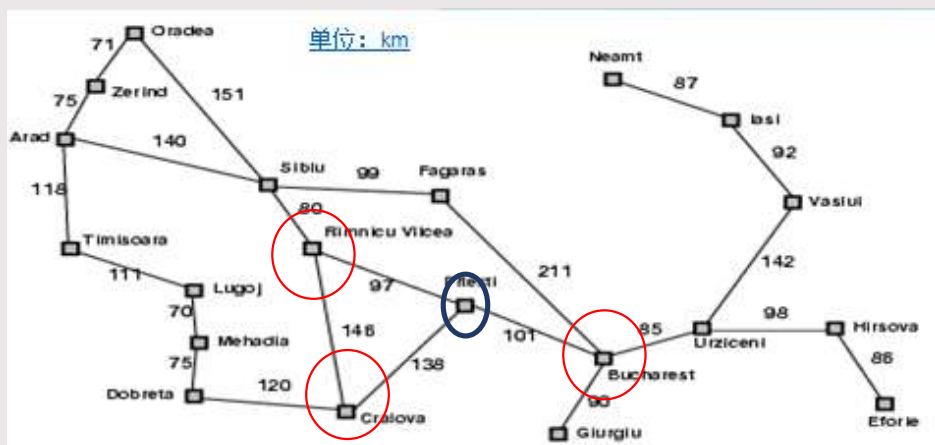
Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374



A*搜索示例



Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374



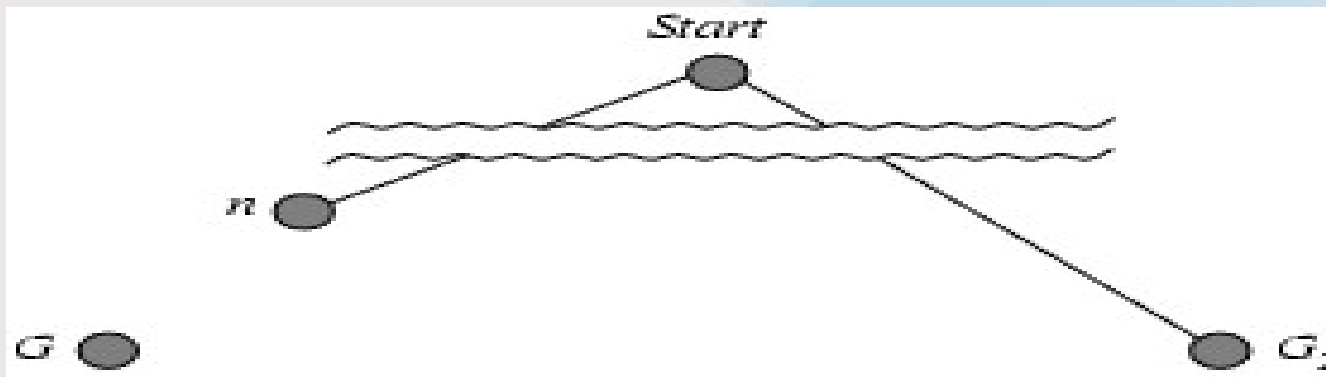
可采纳的启发式函数（admissible heuristic）

- 如果启发式函数 $h(n)$ 对于任意的节点 n 都满足 $h(n) \leq h^*(n)$ ，这里 $h^*(n)$ 是指从节点 n 到达目标的真正代价，则称 $h(n)$ 是可采纳的（admissible heuristic）。
- 定理：如果 $h(n)$ 是可采纳的，则 A^* 树搜索算法是具有最优性。

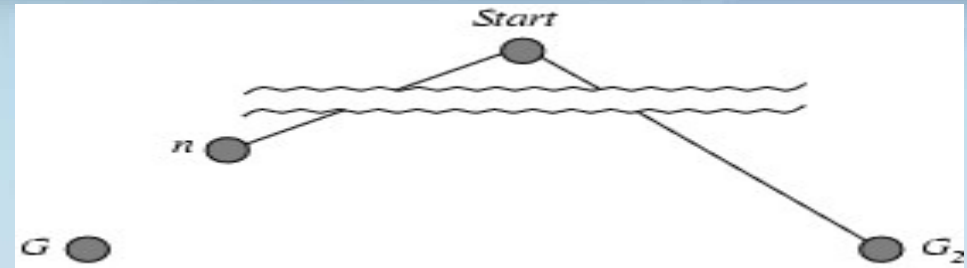


A* 的最优性证明

- 假设某次优目标节点 G_2 已经产生并在 **fringe** 表中排队，设 n 是 **fringe** 表中到达最优目标节点 G 的最短路径上的一个未扩展节点



A* 的最优性



① $f(G_2) = g(G_2)$

② $f(G) = g(G)$

③ $g(G_2) > g(G)$

④ $f(G_2) > f(G)$

⑤ $h(n) \leq h^*(n)$

⑥ $g(n) + h(n) \leq g(n) + h^*(n)$

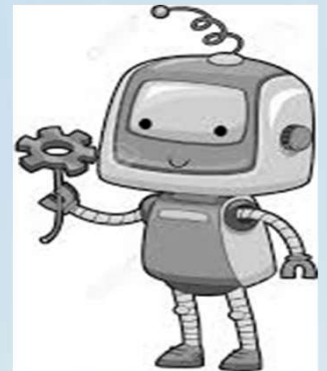
⑦ $f(n) \leq f(G)$

⑧ $f(G_2) > f(n)$

所以 A* 决不会在选择节点 n 之前选择次优节点 G_2 扩展

A*搜索算法的性质

- 完备性? Yes
- 时间? 指数级
- 空间? 将所有产生的节点存储在内存中
- 最优性? Yes



5.3 A*的改进方法

存储受限制的启发式搜索：

思想：迭代加深的A*算法，用一个f-cost代替有限制的深度优先中的d作为截断值，进行剪枝。

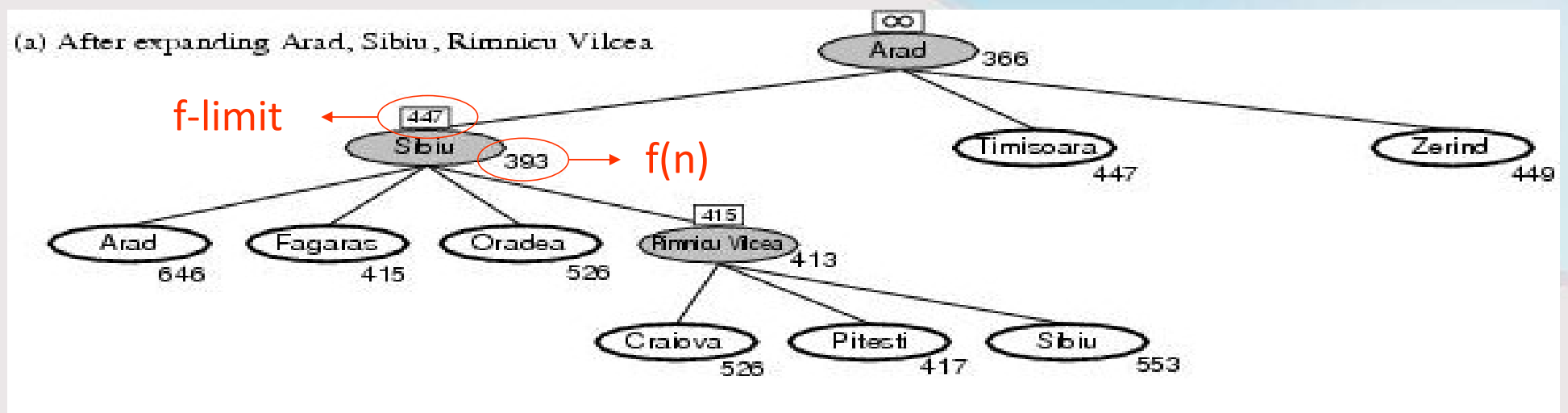
✓ 递归最佳优先搜索（RBFS）

- 尝试模拟标准的最佳优先搜索而只需线性空间。
 1. 记录当前节点的祖先可得到的最佳可替换路径的f值。
 2. 如果当前的f值超过了这个限制，则递归将转回到替换路径。
 3. 向上回溯改变f值到它的孩子的最佳f值
 4. 重复扩展这个上个节点，因为仍有可能存在较优解。

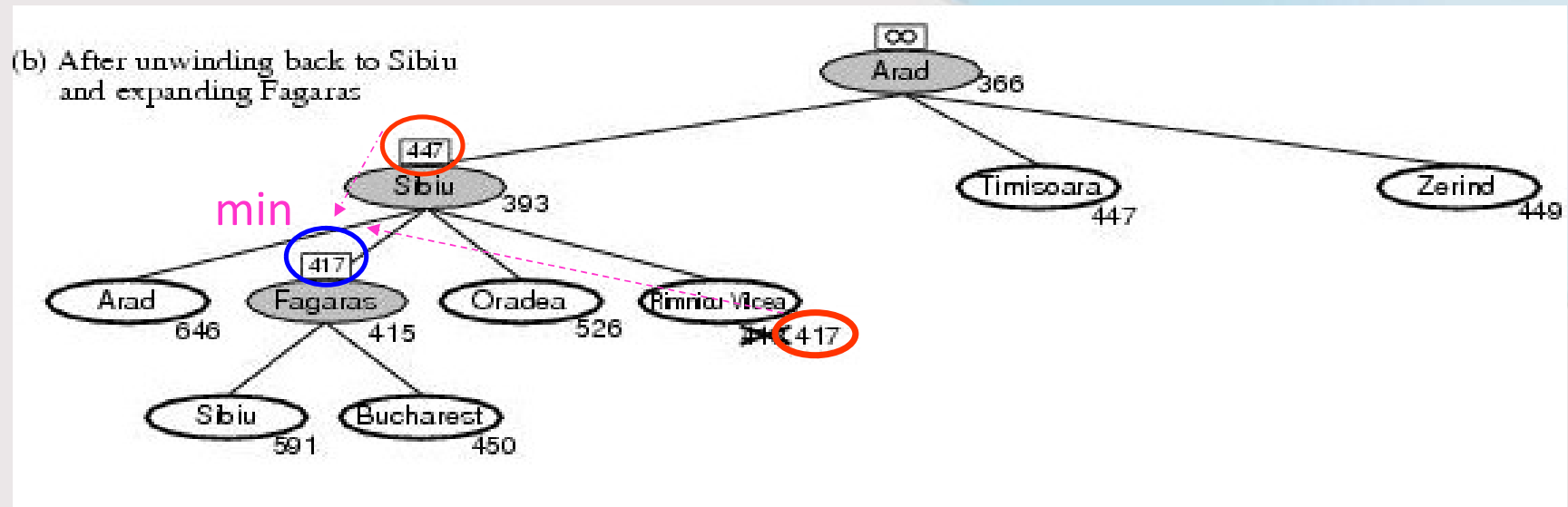
✓ （简单）存储限制的A*

- 当内存满了的时候删除最坏的节点

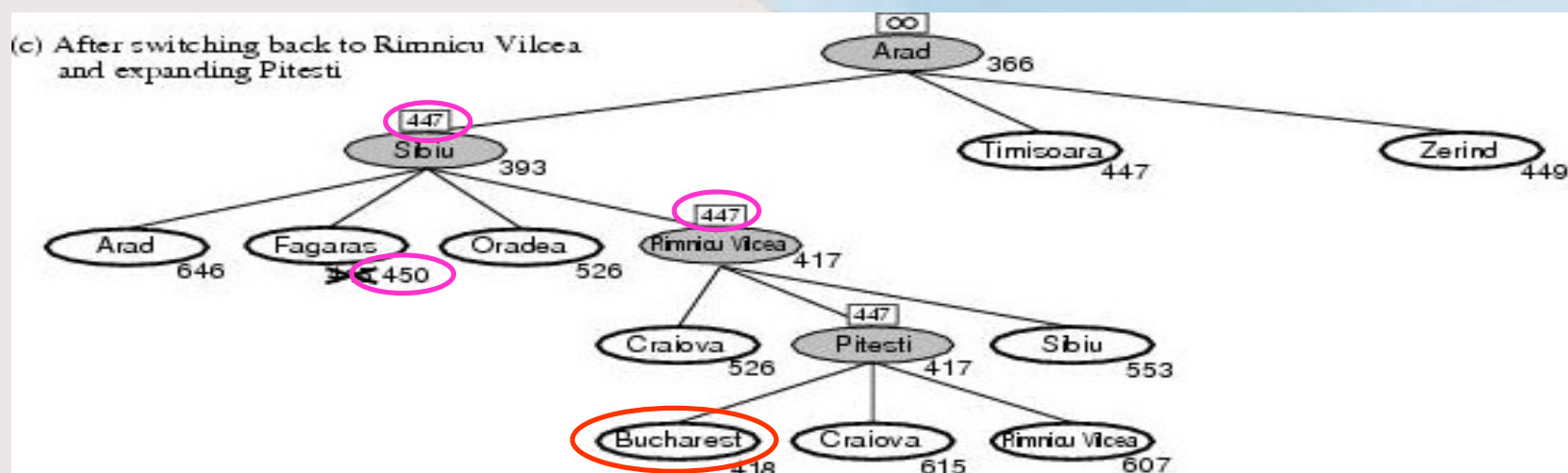
递归最佳优先搜索例子



递归最佳优先搜索例子



递归最佳优先搜索例子



性能:

同A*一样, 如果h是可采纳的, 则有最优性;

空间复杂度却是线性的 $O(bd)$;

时间复杂度难以描述: 取决于h的精确和最佳路径变换的次数。

function RECURSIVE-BEST-FIRST-SEARCH(*problem*) **return** a solution or failure

return RFBS(*problem*, MAKE-NODE(INITIAL-STATE[*problem*]), ∞)

function RFBS(*problem*, *node*, *f_limit*) **return** a solution or failure and a new *f*-cost limit

if GOAL-TEST[*problem*](STATE[*node*]) then **return** *node*

successors \leftarrow EXPAND(*node*, *problem*)

if *successors* is empty then **return** failure, ∞

for each *s* in *successors* do

$f[s] \leftarrow \max(g(s) + h(s), f[*node*])$

repeat

best \leftarrow the lowest *f*-value node in *successors*

if $f[*best*] > f_limit$ then **return** failure, $f[*best*]$

alternative \leftarrow the second lowest *f*-value among *successors*

result, $f[*best*] \leftarrow$ RBFS(*problem*, *best*, min(f_limit , *alternative*))

if *result* \neq failure then **return** *result*

(简化) 存储限制 A*算法

- 思想：利用所有可以用的内存；
 扩展所有最佳节点直到内存满了；
 当内存满了，摘掉最差的那些叶子；
 将该节点值备份到父节点中；

如果所有的节点都有相同的f值怎么办？

某些节点被选出了扩展或被删除；

简化的存储限制通过扩展新节点和删除最老的节点来简化处理。

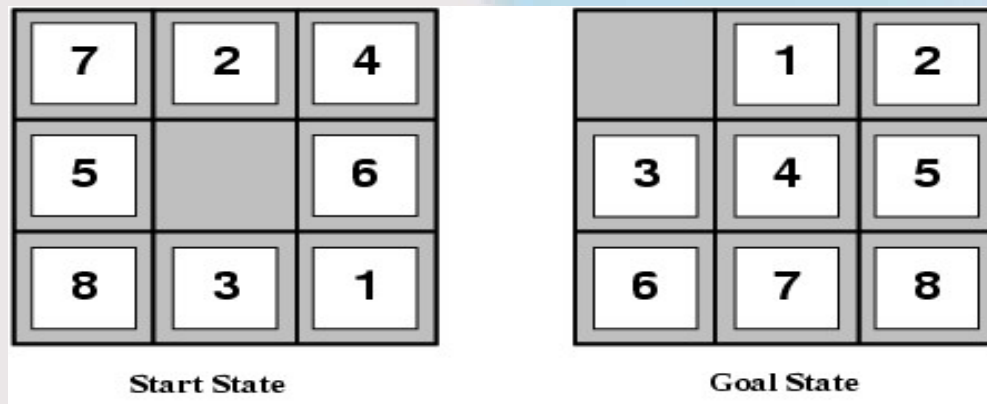
(这是什么思想？深度还是宽度？)

性能：如果有解，简化的存储限制是完备的，也有最优性。

可采纳的启发式函数分析

比如, 对于 8-数码问题:

- $h_1(n)$ = 错放的数字块个数
- $h_2(n)$ = 状态 n 到目标状态的曼哈顿距离
(i.e., 所有数字块到目标位置的曼哈顿距离的和)



可采纳的启发式函数分析

7	2	4
5		6
8	3	1

Start State

	1	2
3	4	5
6	7	8

Goal State

- $h_1(S) = ?$ 8
- $h_2(S) = ?$ $3+1+2+2+2+3+3+2 = 18$

占优势的启发式函数-启发性（控制能力）

- 有效分支因子 b^*
 - 如果A*算法生成的总节点数是N，在解深度为d的一致树中所必须的分支因子 b^* （含有N+1个节点）既有：

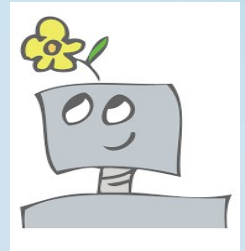
$$N + 1 = 1 + b^* + (b^*)^2 + \dots + (b^*)^d$$

- 对于足够难的问题，该度量值是相当稳定的。
 - 因此，在小规模问题集合上实验测量出的 b^* 值，可以为研究启发式的有效性提供很好的指导。
 - 一个好启发式的 b^* 是接近1的。

占优势的启发式函数-启发性（控制能力）

- 为了测试 h_1 、 h_2 的质量和控制在随机产生了1200个八数码问题，他们的解长度为2到24，分别用迭代深度优先、 $A^*(h_1)$ 、 $A^*(h_2)$ 求解得到 b^* 与实际代价的测试结果如下：

d	Search Cost			Effective Branching Factor		
	IDS	$A^*(h_1)$	$A^*(h_2)$	IDS	$A^*(h_1)$	$A^*(h_2)$
2	10	6	6	2.45	1.79	1.79
4	112	13	12	2.87	1.48	1.45
6	680	20	18	2.73	1.34	1.30
8	6384	39	25	2.80	1.33	1.24
10	47127	93	39	2.79	1.38	1.22
12	3644035	227	73	2.78	1.42	1.24
14	—	539	113	—	1.44	1.23
16	—	1301	211	—	1.45	1.25
18	—	3056	363	—	1.46	1.26
20	—	7276	676	—	1.47	1.27
22	—	18094	1219	—	1.48	1.28
24	—	39135	1641	—	1.48	1.26



- 在两个启发式均为可采纳的情况下：对于所有 $h_2(n) \geq h_1(n)$ 则 h_2 的启发性（控制力）要优于 h_1 。

5.4 如何构建启发函数

- 可采纳的启发式可以源自简化版问题的一个精确解。

称为松弛问题：对原定问题的动作的约束放宽，以松弛化问题的最优解的代价来定义原问题的一个可采纳启发式函数。

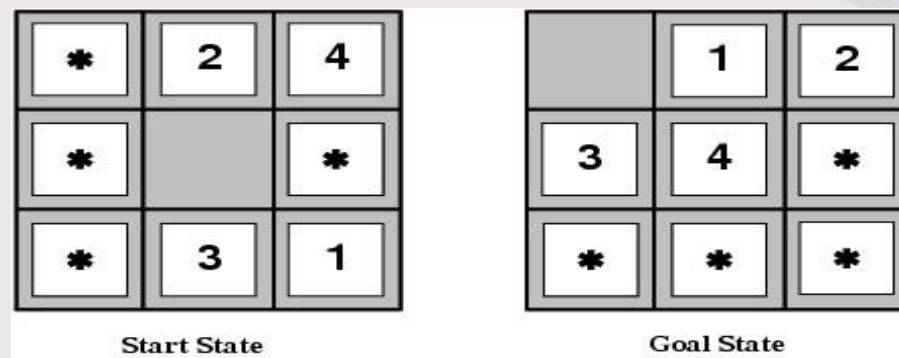
- 1)对于 h_1 来说松弛8数码问题：一个牌可以移到任意的位置。因此， h_1 是这个问题的最短路径解。
- 2)对于 h_2 来说松弛8数码问题：一个牌可以移到任一相邻的位置。因此， h_2 是这个问题的最短解。

明显的，一个松弛问题的最优解一定不会大于真实问题的最优解。

如：**ABSolver** 程序给出了魔方游戏第一个有用的启发式。

如何构建启发函数

- 可采纳的启发式可以从原问题的子问题的一个解得到。
- 这个代价是真实问题代价的下界。
- 将每个可能的子问题实例的精确解存储起来作为一个模式数据库。
 - 用这个模式数据库的构建一个完整的启发式



如何构建启发函数

- 另外一种方法就是从经验中学习：
 - 经验=求解大量的8数码游戏。
 - 例如：8数码的 $f(n)=g(n)+p(n)+3s(n)$ 效率最高。
p:曼哈顿距离 s: 是否跟在正确的牌后

5.5 通过搜索进行问题求解——有信息的搜索

- 最佳优先搜索: $f(n)=g(n)+h(n)$
- 贪婪最佳优先搜索: $f(n)=h(n)$
- A* 搜索: 其中 $h(n) \leq h^*(n)$ 是可采纳的。
- A* 的改进
- 启发函数的启发能力:
- 可采纳启发式设计: 松弛问题 模式数据库 学习机制

作业：编程实现：罗马尼亚度假问题

搜索策略

- 分别采用：宽度优先、深度优先、贪婪算法和A*算法实现。
- 并能比较算法性能。

编程实现

- 编程语言自选
- 输入罗马里亚简化地图

运行结果

- 图形化界面包含：
地图、算法选择、耗散值、生成节点数统计、运行时间、路径搜索动态*示意等

