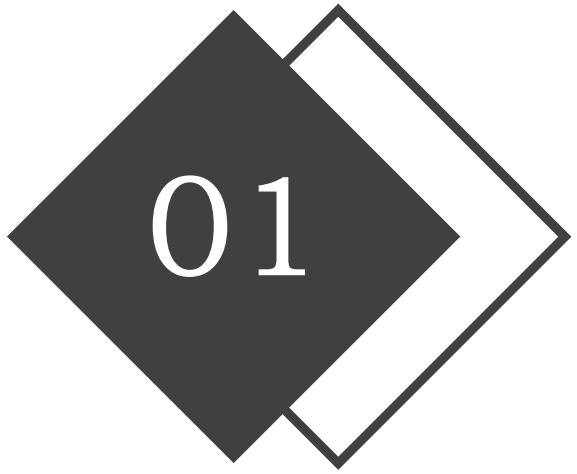
启发式搜索

目录

- 理论课回顾
  - 启发式搜索
  - A*
  - A*的性质
  - IDA*
  - 启发式函数设计
- 实验任务



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

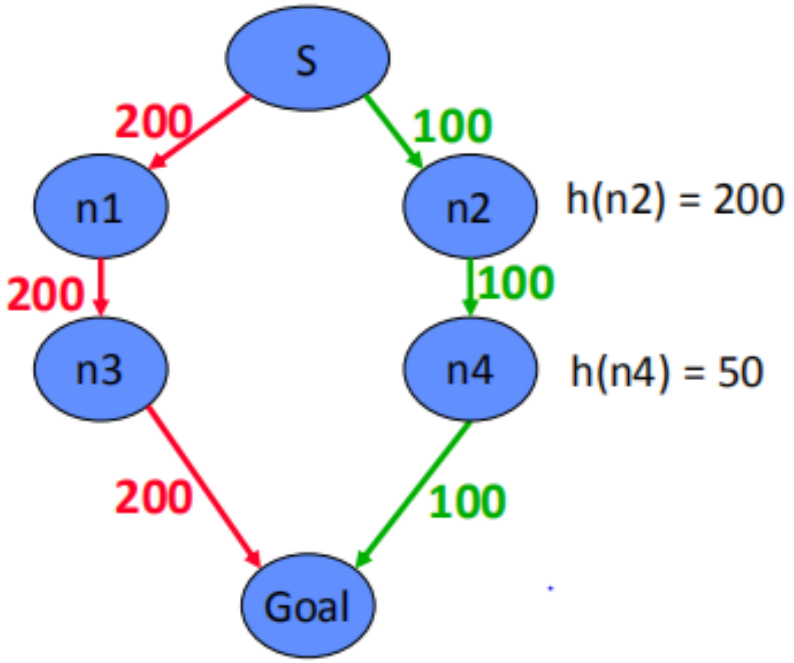
→ step cost = 200
→ step cost = 100

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 A* 搜索算法

输入: 初始状态 s_0, 动作集合, 转移模型 T, 目标检测函数, 启发函数 h

```
初始化列表 open← [s0], close← []
while open 非空 do
    从 open 列表找到 f 值最低的节点 s
    open.remove(s)
    close.append(s)
    if s 是目标节点 then
        return 搜索路径
    end if
    for all s 的相邻节点 s' do
        if s' in close then
            pass
        else if s' in open then
            更新 g(s') 的值
        else
            open.append(s')
        end if
    end for
end while
```

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 IDA* 搜索算法

输入: 初始状态 s_0, 动作集合, 转移模型 T, 目标检测函数, 启发函数 h

初始化深度限制 $c \leftarrow 1$, 栈 stack$\leftarrow[s_0]$, 辅助变量 $c' = \infty$
while True **do**
  if $c' < \infty$ **then**
    $c \leftarrow c', c' \leftarrow \infty$
  end if
  while stack 非空 **do**
    $s \leftarrow$ stack.pop()
    if s 是目标节点 **then**
      return 搜索路径
    else
      for all s 的邻接节点 s' **do**
        if $f(s') \leq c$ **then**
          stack.append(s')
        else
          $c' \leftarrow \min\{c', f(s')\}$
        end if
      end for
    end if
  end while
  if stack 为空且 $c' = \infty$ **then**
    return 搜索失败
  end if
end while

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

➢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）：$D = |x_1 - x_2| + |y_1 - y_2|$

在正方形网格中，允许向8邻域的移动，使用对角线距离（L∞）：$D = \max(|x_1 - x_2|, |y_1 - y_2|)$

启发函数没有限制，大家可以多尝试几种。

第4次作业(第1页/共1页)

1. 实验报告中需包含对所实现算法的原理的解释.
2. 可以假设输入的puzzle问题总是有解的.
3. Puzzle问题的规模$2 \leq n \leq 5$.
4. 可以在代码中定义多种启发式函数, 并在报告中分析不同启发式函数对算法性能的影响. 最终提交的算法(函数)中只需包含性能最好的启发式函数.
5. 请不要import其他库.

1.启发式搜索算法

在A*算法和IDA*算法中选择1种, 解决(n^2-1)-puzzle问题, 可自定义启发式函数. Puzzle问题的输入数据类型为二维嵌套list, 空位置用 `0`表示. 输出的解数据类型为 `list`, 是移动数字方块的次序.

若选择A*算法, 则函数名为 `A_star`; 若选择IDA*算法, 则函数名为 `IDA_star`.

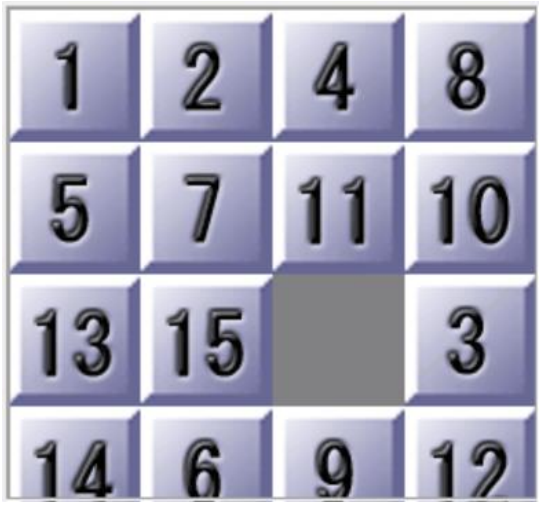
例子: 输入$n=4$规模的问题

```
puzzle = [[1,2,3,4],[5,6,7,8],[9,10,11,12],[0,13,14,15]]
```

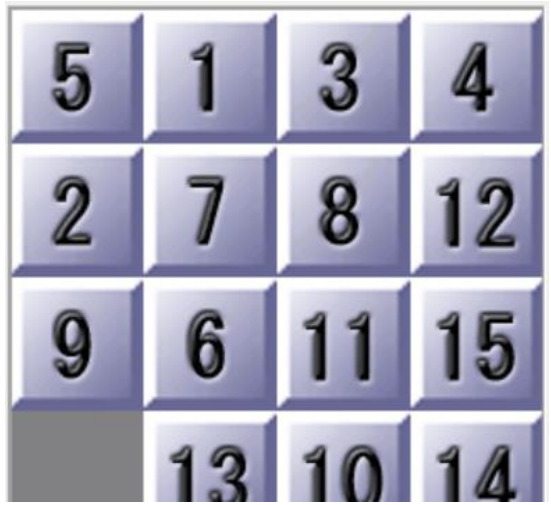
则调用 `A_star(puzzle)`或 `IDA_star(puzzle)`后输出解

```
[13,14,15]
```

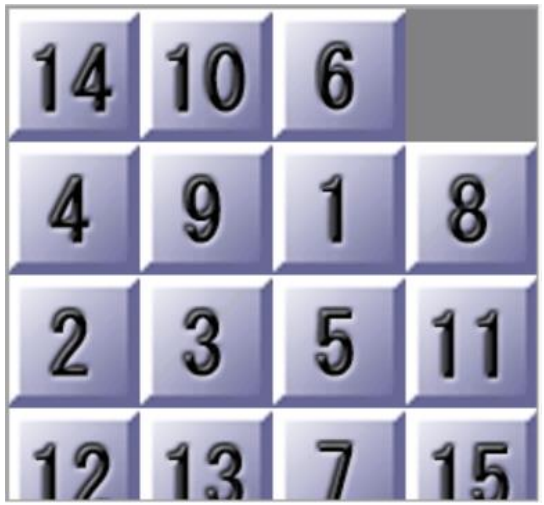
基本样例:



样例1



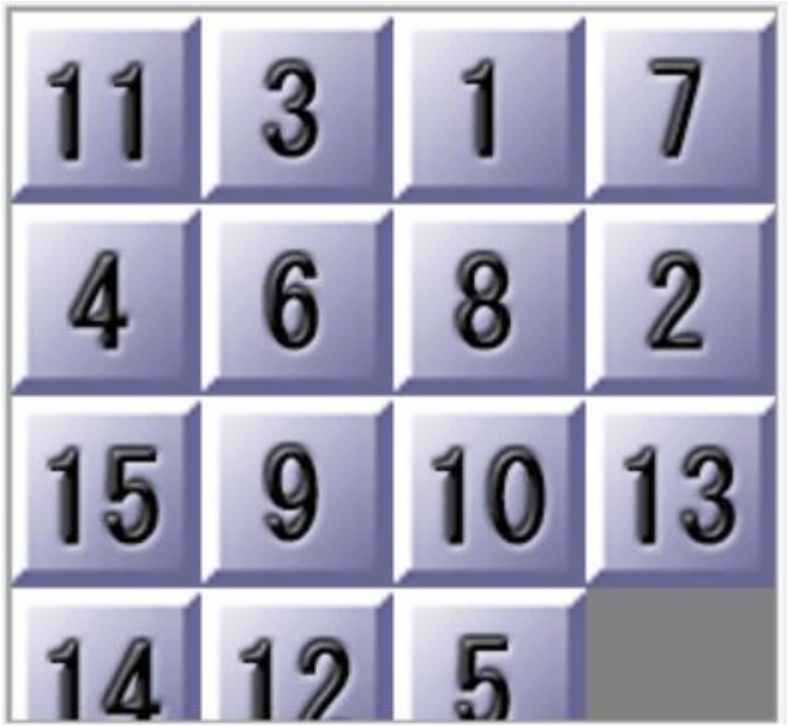
样例2



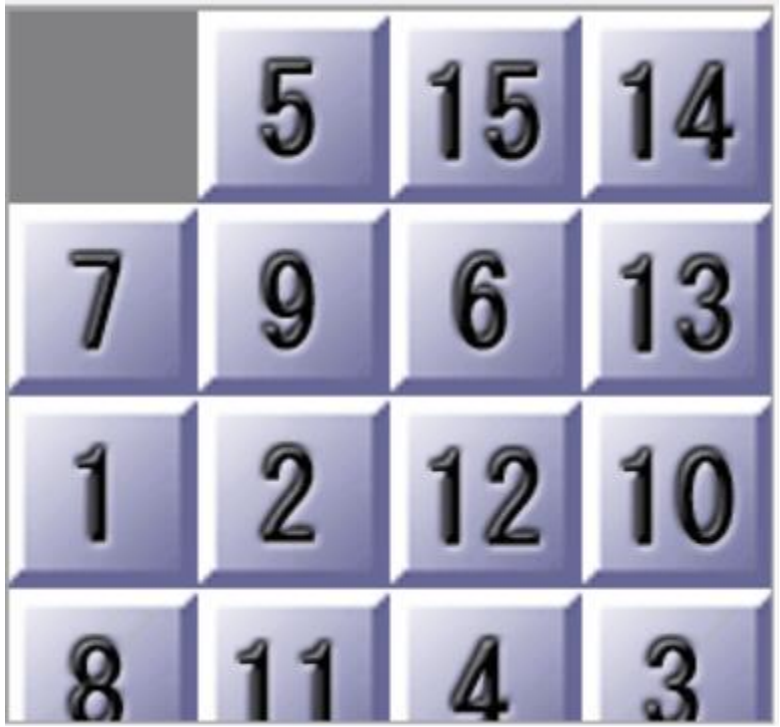
样例3

样例4

挑战性样例:



样例5



样例6