# 算法说明

## 主要思路

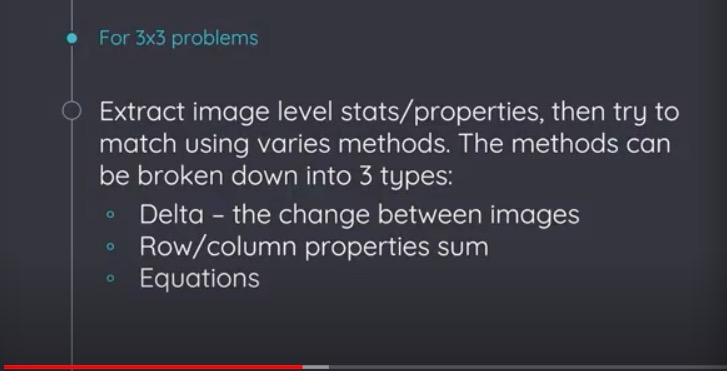
RPM 问题 , 2x2 的图形结构:

|  |  |
| --- | --- |
| A | B |
| C | ? |

3X3 的图形结构

|  |  |  |
| --- | --- | --- |
| A | B | C |
| D | E | F |
| G | H | ? |

需要求解 ? 处的图形, 求解的主要思路是, 归纳出当前给定问题图形中每行,每列,以及对角线上同一组的2个或3个图形的变换规律, 属性等, (如视频



<https://www.youtube.com/watch?v=xcaBWs12WDg> 04:24 位置提到的), 将每个答案替换到?处,使用不同方法将答案所在行/列与其他行/列进行相关属性的匹配,匹配程度越高的答案, 可信度越高. 将不同属性给定一个可信度的分数, 将所有匹配成功的分数值相加, 取 最大值的答案.

目前归纳了下列的一些行/列属性:

1. 2x2 的图形:

|  |  |  |
| --- | --- | --- |
| **行/列 属性** | **说明** | **可信度分数** |
| 相等或相似 | 例如第一行中的AB 两个图形相同(如全为正方形),如果 第二行C? 的两个图形也相同, 该属性匹配成功. 如数据B-01 | 10 |
| 翻转  (垂直,水平) | 例如一行中的AB 两个图形, B图形与A图形垂直或水平翻转后的图形相等,  如数据B-03 | 10 |
| 填充 | 例如一行中的AB 两个图形, A是空心图, 填充后变为实心图与B相等 或 B是空心图, 填充后变为实心图与A相等  如数据B-09 中的 AB, B-07 中的 AC | 10 |
| 旋转  (90,180,270 | 例如一行中的AB 两个图, A 经过 90度,或 180度,或 -90度旋转后,与B相同 | 10 |
| 元素增减规律 | 以 AB / C? 这两行匹配为例子, A中有两个元素,B中有三个元素,如果行C?中C是两个元素,?是三个元素. 该属性匹配成功  如果 B中有两个元素与A中的两个元素相等或相似, 则可信度加大  如果元素同时减少, 也满足该规则  例如: B-10 的问题 | AB中含同元素时分数=10  否则 =3 |
| 其他 | 像素, 图形顶点 等属性, 程序暂时未考虑 |  |

1. 3x3 的图形:

|  |  |  |
| --- | --- | --- |
| **行/列 属性** | **说明** | **可信度分数** |
| 相等或相似 | 例如:第一行中的ABC 三个图形相同(如全为正方形),如果 第三行GHC? 的三个图形也相同, 该属性匹配成功. 如数据C-01 , C-02 | 10 |
| 相同组合 | 例如:第三行GH? 的三个图形包含在第一行中的ABC 三个图形中, 可能次序不同 .  如 Challenge Problem D-02 中的 ABC 与 GH1 | 10 |
| 元素增减规律 | ABC 的三个图形个数按同样规律变化   1. 线性递增 2. 倍数递增 (如 C-03的ABC / GH4 或 GH6 ) | 4  或 5分(元素全相等时) |
| 图形面积大小的变化规律 | 面积大小都按从小->大, 或从大->小的规律变化  例如 C-02 中 ABC/GH4 和 ABC/GH1, 图形都相似, 但ABC/GH4 满足面积按小->大的规律变化,得分要比 ABC/GH1 匹配的得分高 | 2 |
| 旋转 | 例如ABC, B是A的90度旋转,C是B的90度旋转  或 B是A的45度旋转,C是B的45度旋转(如 Challenge Problem D-04) | 10 |
| 按像素AND OR XOR | 例如 E-04 中的 ABC, B与C按像素合并得到 A  (目前程序只考虑了 像素的合并操作, XOR 等未考虑) | 7 |
| 其他 | 图形顶点 等属性, 程序暂时未考虑 |  |

( 上面总结的 属性 只是初步考虑. 属性,分值 后面可能按需要调整, 再加一下其他的 匹配属性 , 目前先实现一个 大体算法 框架)

匹配的 行/列/对角线 , 归纳了下列情况:

1. 2x2 的图形:

|  |  |  |
| --- | --- | --- |
| **比较行/列** | **答案所在行/列** | **说明** |
| AB | C? | 行属性比较 |
| AC | B? | 列属性比较 |
| BC | A? | 对角线属性比较(分值权重较行列低) |

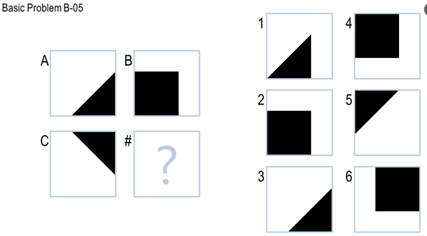
1. 3x3 的图形

|  |  |  |
| --- | --- | --- |
| **比较行/列** | **答案所在行/列** | **说明** |
| ABC | GH? | 第三行与第一行比较 |
| DEF | GH? | 第三行与第二行比较 |
| ADG | CF? | 第三列与第一列比较 |
| BEH | CF? | 第三列与第二列比较 |
| BFG | AE? | 对角线属性比较(分值权重较行列低) |
| CDH | AE? | 对角线属性比较(分值权重较行列低) |

下面用几个具体例子,说明匹配过程

1. 2x2 的例子 :

Basic Problem B-05 ,



程序匹配的得分表:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 匹配  行/列 | 匹配属性 | 答案 | | | | | |
| 1 | 2 | 3 | 4 | 5 | 6 |
| AB-C? | 元素增减规律 | 3 | 3 | 3 | 3 | 3 | 3 |
|  |  |  |  |  |  |  |
| AC-B? | 元素增减规律 | 3 | 3 | 3 | 3 | 3 | 3 |
| 翻转(FLIPV) |  | 10 |  | 13 |  | 10 |
| 得分合计 |  | 6 | 16 | 6 | 19 | 6 | 16 |

(注: 得分==0 的属性 未列出 )

根据 总得分, 选择答案 4

( 答案 4 得分 高 是因为: AC - B4 满足 按整个图片 上下翻转 的 变化规律 , AC - B2 也是正方形, 满足 翻转的条件, 但 不是基于 整个图片翻转 )

程序执行的调试信息:

--->This Problem: Basic Problem B-05

答案 [1] 总得分 = 6.00 , 其中:

得分 3.00 来自于: [AB-C1]两组元素个数匹配,增减个数相同

得分 3.00 来自于: [AC-B1]两组元素个数匹配,增减个数相同

答案 [2] 总得分 = 16.00 , 其中:

得分 3.00 来自于: [AB-C2]两组元素个数匹配,增减个数相同

得分 10.00 来自于: [AC-B2]元素0匹配相同变换FLIPV

得分 3.00 来自于: [AC-B2]两组元素个数匹配,增减个数相同

答案 [3] 总得分 = 6.00 , 其中:

得分 3.00 来自于: [AB-C3]两组元素个数匹配,增减个数相同

得分 3.00 来自于: [AC-B3]两组元素个数匹配,增减个数相同

答案 [4] 总得分 = 19.00 , 其中:

得分 3.00 来自于: [AB-C4]两组元素个数匹配,增减个数相同

得分 13.00 来自于: [AC-B4]元素0匹配相同变换FLIPV(基于整图翻转)

得分 3.00 来自于: [AC-B4]两组元素个数匹配,增减个数相同

答案 [5] 总得分 = 6.00 , 其中:

得分 3.00 来自于: [AB-C5]两组元素个数匹配,增减个数相同

得分 3.00 来自于: [AC-B5]两组元素个数匹配,增减个数相同

答案 [6] 总得分 = 16.00 , 其中:

得分 3.00 来自于: [AB-C6]两组元素个数匹配,增减个数相同

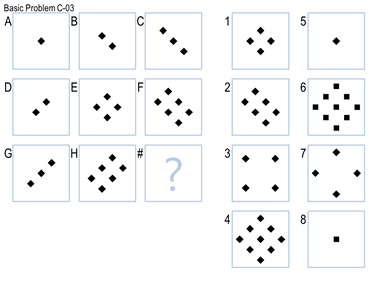
得分 10.00 来自于: [AC-B6]元素0匹配相同变换FLIPV

得分 3.00 来自于: [AC-B6]两组元素个数匹配,增减个数相同

[Basic Problem B-05] 结果 = 4

1. 3x3 的例子 :

Basic Problem C-03



程序匹配的得分表:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 匹配  行/列 | 匹配属性 | 答案 | | | | | | | |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| ABC-GH? | 图形相等 | 10 | 10 | 10 | 10 | 10 |  | 10 |  |
| 元素个数按同倍数递增 |  |  |  | 4 |  | 4 |  |  |
| DEF-GH? | 图形相等 | 10 | 10 | 10 | 10 | 10 |  | 10 |  |
| 元素个数按同倍数递增 |  |  |  | 4 |  | 4 |  |  |
| ADG-CF? | 图形相等 | 10 | 10 | 10 | 10 | 10 |  | 10 |  |
| 元素个数按同倍数递增 |  |  |  | 4 |  | 4 |  |  |
| BEH-CF? | 图形相等 | 10 | 10 | 10 | 10 | 10 |  | 10 |  |
| 元素个数按同倍数递增 |  |  |  | 4 |  | 4 |  |  |
| BFG-AE? | 图形相等 | 7 | 7 | 7 | 7 | 7 |  | 7 |  |
|  |  |  |  |  |  |  |  |  |
| CDH-AE? | 图形相等 | 7 | 7 | 7 | 7 | 7 |  | 7 |  |
|  |  |  |  |  |  |  |  |  |
| 总分 |  | 54 | 54 | 54 | 70 | 54 | 16 | 54 | 0 |

( 注: 属性 “图形相等” 是指公共 元素相等, 例如 ABC 只比较第一个, BEH 比较两个)

程序执行的调试信息:

答案 [1] 总得分 = 54.00 , 其中:

得分 10.00 来自于: [ABC-GH1]每组图形全相等

得分 10.00 来自于: [DEF-GH1]每组图形全相等

得分 10.00 来自于: [ADG-CF1]每组图形全相等

得分 10.00 来自于: [BEH-CF1]每组图形全相等

得分 7.00 来自于: [BFG-AE1]每组图形全相等

得分 7.00 来自于: [CDH-AE1]每组图形全相等

答案 [2] 总得分 = 54.00 , 其中:

得分 10.00 来自于: [ABC-GH2]每组图形全相等

得分 10.00 来自于: [DEF-GH2]每组图形全相等

得分 10.00 来自于: [ADG-CF2]每组图形全相等

得分 10.00 来自于: [BEH-CF2]每组图形全相等

得分 7.00 来自于: [BFG-AE2]每组图形全相等

得分 7.00 来自于: [CDH-AE2]每组图形全相等

答案 [3] 总得分 = 54.00 , 其中:

得分 10.00 来自于: [ABC-GH3]每组图形全相等

得分 10.00 来自于: [DEF-GH3]每组图形全相等

得分 10.00 来自于: [ADG-CF3]每组图形全相等

得分 10.00 来自于: [BEH-CF3]每组图形全相等

得分 7.00 来自于: [BFG-AE3]每组图形全相等

得分 7.00 来自于: [CDH-AE3]每组图形全相等

答案 [4] 总得分 = 70.00 , 其中:

得分 10.00 来自于: [ABC-GH4]每组图形全相等

得分 4.00 来自于: [ABC-GH4]两组图形元素个数变化按同倍数递增

得分 10.00 来自于: [DEF-GH4]每组图形全相等

得分 4.00 来自于: [DEF-GH4]两组图形元素个数变化按同倍数递增

得分 10.00 来自于: [ADG-CF4]每组图形全相等

得分 4.00 来自于: [ADG-CF4]两组图形元素个数变化按同倍数递增

得分 10.00 来自于: [BEH-CF4]每组图形全相等

得分 4.00 来自于: [BEH-CF4]两组图形元素个数变化按同倍数递增

得分 7.00 来自于: [BFG-AE4]每组图形全相等

得分 7.00 来自于: [CDH-AE4]每组图形全相等

答案 [5] 总得分 = 54.00 , 其中:

得分 10.00 来自于: [ABC-GH5]每组图形全相等

得分 10.00 来自于: [DEF-GH5]每组图形全相等

得分 10.00 来自于: [ADG-CF5]每组图形全相等

得分 10.00 来自于: [BEH-CF5]每组图形全相等

得分 7.00 来自于: [BFG-AE5]每组图形全相等

得分 7.00 来自于: [CDH-AE5]每组图形全相等

答案 [6] 总得分 = 16.00 , 其中:

得分 4.00 来自于: [ABC-GH6]两组图形元素个数变化按同倍数递增

得分 4.00 来自于: [DEF-GH6]两组图形元素个数变化按同倍数递增

得分 4.00 来自于: [ADG-CF6]两组图形元素个数变化按同倍数递增

得分 4.00 来自于: [BEH-CF6]两组图形元素个数变化按同倍数递增

答案 [7] 总得分 = 54.00 , 其中:

得分 10.00 来自于: [ABC-GH7]每组图形全相等

得分 10.00 来自于: [DEF-GH7]每组图形全相等

得分 10.00 来自于: [ADG-CF7]每组图形全相等

得分 10.00 来自于: [BEH-CF7]每组图形全相等

得分 7.00 来自于: [BFG-AE7]每组图形全相等

得分 7.00 来自于: [CDH-AE7]每组图形全相等

答案 [8] 总得分 = 0.00

[Basic Problem C-03] 结果 = 4

## 程序大体框架

Agent的核心部分是 对于每个答案, 针对前面列出的所有行列 , 两行或两列直接对每个前面列出的属性进行匹配, 根据是否匹配成功 给对应答案可信度的评分

1. 2x2 的问题

Agent.py 中方法:

calculateImages2MatchScore(self,imgs1Name:str,imgs2Name:str,scoreAddTo:AnswerScore,scoreWeight=1)

对 2x2 图形 的两行 或 两列 进行匹配并评分, 评分结果 累加到 scoreAddTo 对象中.

输入参数:

imgs1Name: 为 "AB","AC" 等

imgs2Name: 为 "B1", "B2",..."C1","C2",...等

输出参数:

scoreAddTo 用于累加 评分结果

calculateImages2MatchScore 的代码实现概括如下:

    imgs1 = self.getImages2(imgs1Name)  # "AB" 等的行/列对象

    imgs2 = self.getImages2(imgs2Name)  # "C?" 等的行/列对象

    if  imgs1的图形 与 imgs2的图形 相等或相似:

         scoreAddTo.score += 10  # 匹配到相等规则, 可信度评分加 10分

    if  imgs1的图形 与 imgs2的图形 都满足翻转关系:

        scoreAddTo.score += 10  # 匹配到翻转规则, 可信度评分加 10分

    if  imgs1的图形 与 imgs2的图形 都满足旋转关系:

        scoreAddTo.score += 10  # 匹配到旋转规则, 可信度评分加 10分

......

( 即: calculateImages2MatchScore 所做的工作. 就是所谓的 “规则堆砌” )

例如 要对答案1 进行评分

调用:

answerScore = AnswerScore(1)

self.calculateImages2MatchScore("AB","C1",answerScore)

self.calculateImages2MatchScore("AC","B1",answerScore)

 #  self.calculateImages2MatchScore("BC","A1",answerScore,0.5)

( 注: 对角线现在程序未处理)

程序执行之后, 得到评分结果累加在对象 answerScore 的 score 中, 得分的详细信息 记录在 answerScore.scoreDetails 列表中, 用于执行结果 调试时查看(参看上面例子打印的调试信息)

Agent.py 中 方法:

solve\_2x2(self):

就是对 答案 1,2,3,4,5,6 调用上面的代码, 得到每个答案的 评分, 选取最大值评分的答案返回

1. 3x3 的问题

Agent.py 中, 方法:

calculateImages3MatchScore(self,imgs1Name:str,imgs2Name:str,scoreAddTo:AnswerScore,scoreWeight=1)

对 3x3 图形 的两行 或 两列 进行匹配并评分, 评分结果 累加到 scoreAddTo 对象中

输入参数

imgs1Name 为 "ABC","DEF", "ADG","BEH" 等

imgs2Name 为 "GH1", "GH2",..."CF1","CF2",...等

输出参数:

scoreAddTo 用于累加 评分结果

calculateImages3MatchScore 的代码实现类似于calculateImages2MatchScore , 单 使用的匹配规则不同

例如 要对答案1 进行评分

调用:

answerScore = AnswerScore(1)

 # 行比较 : 第一行 与 第 三 行

self.calculateImages3MatchScore("ABC","GH1",answerScore)

 # 行比较 : 第二行 与 第 三 行

self.calculateImages3MatchScore("DEF","GH1",answerScore)

 # 列比较 : 第一列 与 第 三 列

self.calculateImages3MatchScore("ADG","CF1",answerScore)

 # 列比较 : 第二列 与 第 三 列

self.calculateImages3MatchScore("BEH","CF1",answerScore)

 # 对角线 , 得分权重 0.7 低于 行列的 权重 1

self.calculateImages3MatchScore("BFG","AE1",answerScore,0.7)            self.calculateImages3MatchScore("CDH","AE1",answerScore,0.7)

, 程序执行之后, 得到评分结果累加到对象 answerScore 中

Agent.py 中方法

solve\_3x3(self):

就是对 答案 1,2,3,4,5,6,7,8 调用上面的代码, 得到每个答案的 评分, 选取最大值评分的答案返回

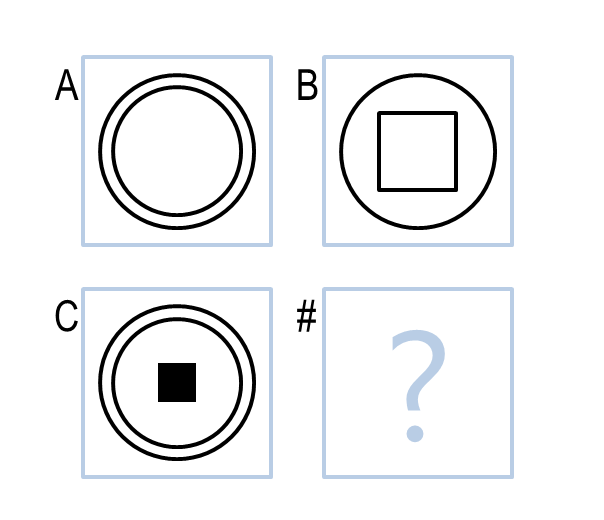
## 关键几个算法与数据结构

了解了上面的程序的主要框架后, 现在剩下的工作就是 calculateImages2MatchScore 和 calculateImages3MatchScore 方法中要做的行列属性的匹配(或叫规则的判断), 如判断两个图形是否相等, 是否满足旋转关系等等. 这里介绍下这两个方法中涉及到关键的算法和数据结构

1. **图形元素的拆分**

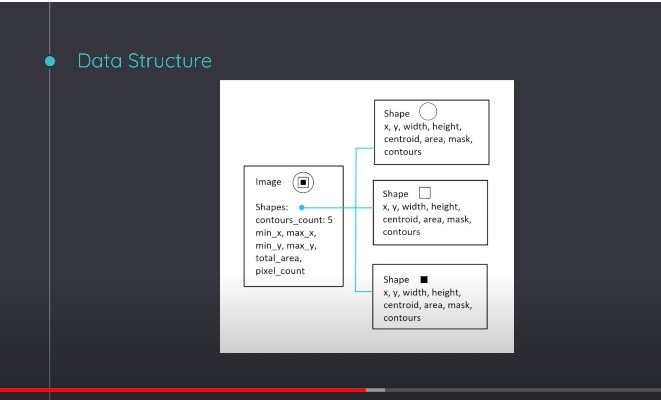
在算法中进行 行列 的属性匹配时, 都是针对图形中的元素的, 特别是 图形元素增减的规律, 更需要图形拆分出元素,才能知道图形中元素的数量

例如 : B-10:

需要拆分出 园,正方行

拆分的原理是: 对于 像素相连的 , 在一个 元素中, 否则 在不同元素中

( 视频的 05:20 提到了 拆分的 数据结构



)

Agent.py 中 拆分元素的数据结构由

class ImageElement

类定义,

拆分方法在 ImageElement的静态方法

ImageElement.splitImage(image)

中, 拆分具体实现细节算法暂时不展开, 如果项目需要使用再详细补充到文档

(PS : 网上有些 实现 基于

<https://github.com/spwhitt/cclabel/blob/master/cclabel.py>

的方法实现, 感觉 这个方法效率要高些,

### cv2 的 [floodFill](http://www.baidu.com/link?url=5-3NvofTSHZXa_BzmagVYjS-Jh-6_Cgri4o_MNwTynoPMEoG-atXBHOkXV9yYaz6V46pfNZyclm8onjivwGe8uGo_Zfq9fkeWv5EfD6EgV7" \t "https://www.baidu.com/_blank) 是否可用在拆分 算法中 还有待 研究

目前 拆分的 结果中 还不能 含 图形 顶点 信息, 所以 比较属性中 现在没有 顶点 个数的比较

)

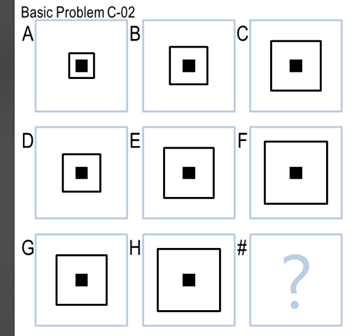
1. **图形元素相同的判断**

在很多 行/列 属性比较时, 最终都涉及到 两元素 是否为同一个元素的问题(相等 或 相似), 例如 判断图形旋转后是否为同一图形.

原来代码中有一个简单的方法,

distance = np.sqrt(np.sum(np.power(p1 - p2, 2)))

根据两个图片的像素 差异 来判断, 但具体应用 还涉及图形相似的比较, 例如都是 正方形, 但大小不同:

****

在比较AB 是否 相等/相似 时, 无法根据 像素 判断, 因为 图形已经被 放大. 所以代码重写了 判断的方法

ImageElement 对象中方法:

getImageElementSimilarScale(self,otherImgElement)

判断 self 是否 与 otherImgElement 两个元素 相等或相似, 实现细节的文档看需要再补充

1. **2x2 行/列 对象 Images2**

类

class Imagegs2

用于定义 2x2 图形的 一行 或一列 两个图片之间的关系及操作, 例如 行 AB

通过 构造对象 Imagegs2 (agent,"AB") 来描述, AB 之间相关属性的判断 定义在 Imagegs2 类中, agent 的 getImagegs2 ("AB") 可以获取一个被缓存的Imagegs2 对象

例如 判断 AB 是否满足相等的属性,通过调用

self.getImagegs2 ("AB").isImgElementTransMatched(IMGTRANSMODE\_EQ)

判断

判断 AB 是否满足垂直翻转的属性, 通过调用

self.getImagegs2 ("AB").isImgElementTransMatched(IMGTRANSMODE\_FLIPV)

判断

判断 AB 是否满足旋转90度的属性, 通过调用

self.getImagegs2 ("AB").isImgElementTransMatched(IMGTRANSMODE\_ROTATE1

)

判断,

等等

在 3x3 的算法中, 也会涉及到两个图片之间相关属性的判断, 同样是使用Imagegs2 来判断, 同样可以 获取 self.getImagegs2(“AB”) 进行 AB 两个图片之间属性的判断 .

1. **3x3 行/列 对象 Images3**

类似于Imagegs2 ,

class Imagegs3

用于定义3x3 图形的 一行 或一列 三个图片之间的关系及操作,例如 行 ABC

通过 构造对象 Imagegs3 (agent,"ABC") 来描述图形的 一行

agent 的 getImagegs3 ("ABC") 可以获取一个被缓存的Imagegs2 对象

1. **类** AnswerScore

class AnswerScore

记录某个答案 的 评分信息

1. **缓存的问题(效率的考虑)**

在进行属性匹配有某些操作时, 有些操作可能是费时的, 例如有些操作可能需要对每个像素的循环比较等, 而一个操作在比较每个答案都可能执行到, 例如 比较行 的旋转90度属性判断, 对答案1,要执行 AB/C1 之间匹配的判断,需要判断 B 是否是 A 的 90度旋转, 对答案1,2,3, ... AB 之间的旋转关系都要被判断,而旋转操作可能是费时的, 所以要将 以前的操作结果 缓存

1. **尽量减少不必要的操作(效率的考虑)**

例如在做 旋转操作前, 先判断两个图片的 像素个数是否相差不大, 否则没必要做, 因为 旋转后的图片 与原图片 不会有较大的像素个数的差异

1. **其他算法**

有关 旋转, 翻转 等,使用cv2的方法实现, 其他还有些空心图填充等一些算法

## 运行测试代码

#### 实现的 Agent.py 在 <https://github.com/caoyuwu/Temp/tree/main/RPM> 目录下, 如果要针对某一个问题求解, 可以下载该目录下的的 Agent.py, TestAgent.py 和 CV2Utils.py , 下载到一个目录下, 再 补上 其他 必要的文件 (RavensProblem.py 等, ) 及 数据 Problems 目录下的文件, 比如 要 调试 B-05 的求解, TestAgent.py 的 main 方法 注解掉其他行, 保留:

def main():

    Agent.\_DEBUG = True

#...

testAgentSolve("B-05")

然后 在该目录下执行

python TestAgent.py

会 打印一些 B-05 求解的得分信息.

TestAgent.py 也有一些其他方法的 调试程序, 例如 调试图形的拆分,

将 main 代码 改成 :

def main():

    Agent.\_DEBUG = True

#...

showSplitImages("B-10","C")

可以模拟打印问题 B-10 中 图片 C的拆分的结果.