**حل مسئله مکعب روبیک با الگوریتم گروه میگوها KHA در پایتون**

**صورت مسئله:**

یک مکعب 6 وجهی که هر وجه آن 3x3 است داریم که 6 رنگ موجود است و از هر رنگ 9 تا وجود دارد مکعب استاندارد روبیک دارای رنگ بندی زیر است: زرد در بالا ، سفید در پایین ، سبز در جلو ، آبی در پشت ، قرمز در سمت چپ و نارنجی در سمت راست است که هدف ما برای حل مسیله هم رسیدن به چنین حالتی است.

yyyyyyyyyoooooooooggggggggggwwwwwwwrrrrrrrrrrbbbbbbbbb

مقدار اولیه که حالت جواب هم به همین صورت زیر است:

brbwygybwryrrobgyogwggggwrobbygwowwbwrrbroowryyoyboyog

b آبی، r قرمز،w سفید،y زرد،g سبز،o نارنجی می باشد. که از هر کدام 9 تا وجود دارد و هر وجه باید به یکی از این رنگ ها دراید. 6 کارکتر اول وجه بالا و بقیه به ترتیب راست و جلو و پایین و چپ و عقب را نشان می دهند.

مثلا brbwyg وجه بالایی را نشان می دهد.

برای بهم ریختن مکعب

1.1 برای بهم ریختن مکعب تصادفی الگوریتم R را تایپ کنید.

1.2 این یک الگوریتم را به صورت تصادفی ایجاد بهم ریخته می کند.

1.3 اندازه بهم ریختن مکعب می تواند در هر مکانی بین 5 تا 30 حرکت باشد.

1.4. برای تغییر اندازه بهم ریختن مکعب می توانید به پرونده scrambleGenerator.py بروید و به خط شماره بروید. 11. در کد slen = random.randint (5،30) می توانید دو عدد ارائه دهید و الگوریتم اندازه بهم ریختگی مکعب را در محدوده ارائه شده ایجاد می کند.

برای بهم ریختن مکعب دستی

2.1 برای دادن بهم ریختن مکعب دستی به مکعب از نوع M استفاده کنید.

2.2. هنگام بهم ریختن مکعب دستی کاربر باید دستی تایپ کند.

2.3 حروف مربوط باید بزرگ باشد و بین هر حرکت فاصله باشد.

2.4 حروف معتبر R برای سمت راست ، L برای سمت چپ ، U برای بالا ، D برای پایین ، F برای جلو و B برای عقب هستند.

2.5 حرف می تواند یک آپوستروف (') در کنار آن داشته باشد که می گوید آن جهت را در جهت عقربه ساعت گردانید و حروف با شماره 2 در کنار آن نشانگر این است که ، طرف باید دو بار چرخانده شود. این باید در کنار حرف بدون فاصله ارائه شود.

اندازه بهم ریختن مکعب معتبر و نامعتبر.

3.1 نمونه ای از بهم ریختن مکعب های معتبر: R 'L D2 U F B2 R L2 U2

3.2 بهم ریختن مکعب نامعتبر:

3.2.1. RL - فاصله ای بین هر حرکت وجود ندارد. باید بین هر حرکت فاصله باشد.

3.2.2. r l - حروف کوچک هستند.

3.2.3. R '- این نامعتبر است زیرا بین حرف و ' فاصله دارد. نباید فاصله ای بین باشد. روش صحیح برای نوشتن آن R' است

3.2.4. R 2 - این نامعتبر است زیرا بین حرف و عدد 2 فاصله وجود دارد. نباید بین حرف و عدد 2 فاصله باشد. روش صحیح نوشتن آن R2 است

**جهت گیری مکعب**

مکعب استاندارد روبیک دارای رنگ بندی زیر است: زرد در بالا ، سفید در پایین ، سبز در جلو ، آبی در پشت ، قرمز در سمت چپ و نارنجی در سمت راست.

بنابراین در کد مکعب با جهت گیری زیر درهم می رود: زرد در بالا و سبز در جلو.

هنگامی که الگوریتم در حال اجرا است ، جهت مکعب را در طی هر تکرار برای هر ذره تولید می کند. جهت گیری نشان دهنده محل قرارگیری برچسب های مکعب است.

جهت گیری خارج از برنامه یک رشته است. مکعب حل شده دارای جهت گیری زیر است "yyyyyyyyyoooooooooggggggggggwwwwwwwrrrrrrrrrrbbbbbbbbb".

9 کاراکتر اول نشان دهنده رنگها در بالای مکعب است که زرد است ، "yyyyyyyyy".

9 کاراکتر بعدی نشان دهنده رنگ سمت راست مکعب است که نارنجی است ، ooooooooo.

9 حرف بعدی مربوط به قسمت جلویی مکعب است که سبز است ، "ggggggggg".

سپس 9 بعدی برای سمت پایین است که "wwwwwwwww" سفید است.

سپس 9 بعدی برای سمت چپ است که قرمز است ، "rrrrrrrrrr".

و سرانجام 9 نویسه آخر نشان دهنده رنگهای قسمت پشتی است که آبی است ، "bbbbbbbbbb".

رنگ ها به ترتیب بالا ، راست ، جلو ، پایین ، چپ ، عقب خوانده می شوند.

اگر مکعب را باز کنیم ، جهت زیر را با U برای بالا ، L برای چپ ، F برای جلو ، R برای راست ، B برای عقب و D برای پایین بدست خواهیم آورد.

| U | U | U |

    | U | U | U |

    | U | U | U |

| L | L | L | F | F | F | R | R | R | B | B | B |

| L | L | L | F | F | F | R | R | R | B | B | B |

| L | L | L | F | F | F | R | R | R | B | B | B |

    | D | D | D |

    | D | D | D |

    | D | D | D |

و اگر رنگ زرد در بالا و سبز در جلو باشد ، جهت گیری زیر را خواهیم داشت

    | y | y | y |

    | y | y | y |

    | y | y | y |

| r | r | r | g | g | g | o | o | o | b | b | b |

| r | r | r | g | g | g | o | o | o | b | b | b |

| r | r | r | g | g | g | o | o | o | b | b | b |

    | w | w | w |

    | w | w | w |

    w | w | w | |

و می توانیم از اعداد 1 تا 9 استفاده کنیم که ترتیب خواندن رنگ برای هر طرف مکعب را نشان می دهد

     | 1 | 2 | 3 |

      | 4 | 5 | 6 |

     | 7 | 8 | 9 |

| 1 | 2 | 3 | | 1 | 2 | 3 | | 1 | 2 | 3 | | 1 | 2 | 3 |

| 4 | 5 | 6 | | 4 | 5 | 6 | | 4 | 5 | 6 | | 4 | 5 | 6 |

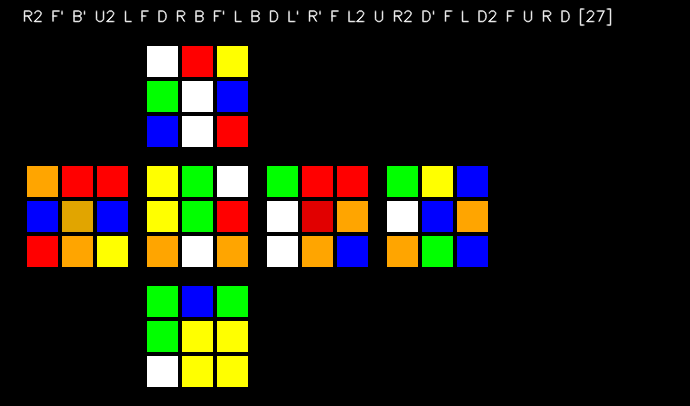
| 7 | 8 | 9 | | 7 | 8 | 9 | | 7 | 8 | 9 | | 7 | 8 | 9 |

    | 1 | 2 | 3 |

    | 4 | 5 | 6 |

| 7 | 8 | 9 |

ابتدا رنگ های سمت بالا ، سپس سمت راست ، سپس قسمت جلو ، سپس سمت پایین ، سپس سمت چپ و در آخر سمت عقب را می خواندیم.



**شرح کد:**

**ارائه بهم ریختگی به مکعب**

وقتی الگوریتم KH را اجرا می کنیم باید مکعب را مخلوط کرده و به الگوریتم بدهیم تا بتواند آن را حل کند. بهم ریختن مکعب را می توان به صورت دستی تهیه کرد یا به صورت تصادفی در برنامه ایجاد کرد. هر برنامه از کاربر می خواهد تا M را برای تهیه یک بهم ریختن مکعب دستی یا نوع R را برای ارائه تقابل تصادفی تایپ کند.

**اجرای کد**

برای اجرای کد برای KH فقط python 4\_KHO.py را اجرا کنید

وقتی برنامه را برای اولین بار اجرا می کنید ، برنامه پایگاه داده ایجاد می کند و جداول را جستجو می کند. این ممکن است یک ساعت یا بیشتر اندازه بکشد. این پایگاه های داده و جداول جستجو شامل تنظیمات متفاوتی از مکعب است که توسط الگوریتم Kociemba برای حل آن استفاده می شود.

الگوریتم Kociemba در اینجا برای یافتن برازندگی مکعب استفاده می شود ، یعنی تعداد حرکات لازم برای حل مکعب از حالت فعلی.

from google.colab import drive

drive.mount('/content/derive')

برای اجرای کد در گوگل کولب و استفاده از گوگل درایو نیازمند اجرای کد بالا هستیم

!pip install -U mxnet-cu101==1.7.0

چون این الگوریتم خیلی زمانبر است از gpu گوگل برای اجرای آن استفاده کردم

!nvidia-smi

import sys

sys.path.append('/content/derive/MyDrive/Code')

کد های پایتون که در گوگل درایو قرار داده بودم را باید در برنامه import کنم از کد بالا برای این کار استفاده می کنم

4\_KHO.py

import FitnessFunction as fit

import Scramble

import math

import random

from scrambleGenerator import makeMove

import matplotlib.pyplot as plt

from os import system, name

import time

import sys

from itertools import groupby

def convert2Scramble(scramble):

  """

  This will take a string "F F' F2"and will convert it into the form [['F',''],['F',"'"],['F','2']].

  We need to give it in the above format for the scrambling to work

  """

  # برای ذخیره حرکات ، یک آرایه خالی ایجاد کنید

  properScramble = []

  # تقسیم بر اساس فاصله بین کاراکترها

  moves = scramble.split(" ")

  #print("Moves : ",moves)

  # آخرین عنصر را به عنوان یک فاصله پاک کنید

  del moves[-1]

  # هر حرکت را انجام دهید

  for m in moves:

    subMoves = ["", ""]

    if m == "F'" or m == "B'" or m == "R'" or m == "L'" or m == "U'" or m == "D'" or m == "F2" or m == "B2" or m == "R2" or m == "L2" or m == "U2" or m == "D2":

      subMoves[0] = m[0]

      subMoves[1] = m[1]

    else:

      subMoves[0] = m

      subMoves[1] = ""

    properScramble.append(subMoves)

  # قالب مناسب برای بهم ریختن مکعب را برگردانید

  return properScramble

# این تابع می خواهد یال را برای یک میگو برگرداند

def ReturnMove(legal\_moves,krill\_AllMoves,krill\_Orientation,krill\_fitness,krill):

  # حالا ما لبه را بر اساس دو شرط انتخاب می کنیم

# ما آستانه را به عنوان فیتنس وضعیت فعلی تنظیم می کنیم

# اگر فیتنس هر یک از لبه های بعدی کمتر از آستانه است ، ما آن لبه را انتخاب می کنیم (استراتژی حریصانه)

# اگر برازندگی تمام لبه بزرگتر یا برابر آستانه باشد ، ما لبه را بر اساس احتمال انتخاب می کنیم

# آستانه را به عنوان برازندگی فعلی میگو تنظیم کنید

  threshold = krill\_fitness[krill]

# اکنون بررسی خواهیم کرد که آیا هر یک از لبه ها از فیتنس کمتری برخوردار است

# ما این پرچم را روی false تنظیم خواهیم کرد

# اگر هر لبه دارای فیتنس کمتر از آستانه باشد ، این پرچم را روی true تنظیم می کنیم

  greedy = False

  move = ""

# این متغیر برای ذخیره حرکت اعمال شده و در صورت یافتن آن بازگرداندن برای آن استفاده می شود

  for m in range(len(legal\_moves)):

    # # حرکت را به مکعب اعمال کنید

    properScramble = convert2Scramble(krill\_AllMoves[krill] + legal\_moves[m] + " ")

    # اندازه بهم ریختگی مکعب را پیدا کنید

    noofMmoves = len(properScramble)

    #  بهم ریختگی مکعب را انجام دهید و جهت گیری کنید

    scramble, cube\_orientation = Scramble.scramble(True, properScramble, noofMmoves)

    # اکنون مقدار فیتنس جهت گیری مکعب را پیدا کنید

    fitnessValue = fit.fitness(cube\_orientation)

    # بررسی کنید کمتر از آستانه است یا حالت حل شده است

    if fitnessValue < threshold or cube\_orientation == "yyyyyyyyyooooooooogggggggggwwwwwwwwwrrrrrrrrrbbbbbbbbb":

  # پرچم حریصانه را به true تنظیم کنید

      greedy = True

      move = legal\_moves[m]

      fv = fitnessValue

# توجه: ما قرار نیست از حلقه خارج شویم و بلافاصله حرکت را برگردانیم

# ما اجازه می دهیم این حلقه تا پایان اجرا شود ، در صورتی که حرکت دیگری پیدا کنیم که فیتنس آن کمتر از حرکت فعلی باشد

# اکنون حرکتی با فیتنس کمتر از آستانه بررسی کنید یا خیر ، بررسی کنید که آیا شاخ حریص تنظیم شده است یا خیر

  if greedy == True:

# روش استفاده شده را چاپ کنید

    print("Method : Greedy",end=" ")

    # اگر درست است ، حرکت و مقدار فیتنس را برگردانید

# زیرا می تواند ارزش فیتنس چندگانه ای برای همان حالت داشته باشد

    return move, fv

# اگر درست نباشد ، باید حرکت بعدی را بر اساس احتمال انتخاب کنیم

# این شبیه ACO است ، اما در اینجا ما از فرمون و همه استفاده نمی کنیم

# همچنین از آنجا که بر اساس احتمال است ، ما در هنگام نزدیک شدن حالت حل شده ، قصد نداریم جزئیات بدهیم.

# به عنوان روش احتمال ، به هر لبه فرصتی می دهد.

  else:

# روش استفاده شده را چاپ کنید

    print("Method : Probability",end=" ")

# اختصاص دادن احتمال به هر حرکت

# در ابتدا مقادیر احتمال برای هر حرکت 0 است

    probabilityDict = {"U":0,"U'":0,"U2":0,"D":0,"D'":0,"D2":0,"L":0,"L'":0,"L2":0,"R":0,"R'":0,"R2":0,"F":0,"F'":0,"F2":0,"B":0,"B'":0,"B2":0}

# این فرهنگ لغت فیتنس را برای هر حرکت ذخیره می کند

# ما برای برگرداندن مقدار فیتنس حرکتی که انتخاب کرده ایم از این فرهنگ لغت استفاده خواهیم کرد

# به این دلیل که یک حالت یکسان مقادیر فیتنس متعددی خواهد داشت ، بنابراین ما باید کمترین ارزیابی را برای همان حالت پیدا کنیم

    fitnessDict = {"U":0,"U'":0,"U2":0,"D":0,"D'":0,"D2":0,"L":0,"L'":0,"L2":0,"R":0,"R'":0,"R2":0,"F":0,"F'":0,"F2":0,"B":0,"B'":0,"B2":0}

# احتمال وجود لبه ها برای اولین بار در bytearay داده می شود

# P (i، j) = (1 / فیتنس\_حرکت بعدی) / (مجموع (فیتنس\_همه لبه های مجاور))

# اکنون تمام حرکات ممکن را انجام دهید

    for m in range(len(legal\_moves)):

     # حرکت را به مکعب اعمال کنید

      properScramble = convert2Scramble(krill\_AllMoves[krill] + legal\_moves[m] + " ")

 # اندازه بهم ریختن مکعب را پیدا کنید

      noofMmoves = len(properScramble)

# آن بهم ریختن مکعب را انجام دهید و جهت گیری کنید

      scramble, cube\_orientation = Scramble.scramble(True, properScramble, noofMmoves)

# اکنون مقدار فیتنس جهت گیری مکعب را پیدا کنید

      fitnessValue = fit.fitness(cube\_orientation)

# فیتنس حرکت را در فرهنگ لغت فیتنس ذخیره کنید

      fitnessDict[legal\_moves[m]] = fitnessValue

# اکنون مقدار eta در شماره گیر (کیفیت لبه = 1 / Lk یعنی معکوس فاصله) است

      etaNumerator = (1/fitnessValue)

# چاپ ("\ n")

# چاپ ("حرکت:" ، حرکت\_ قانونی [متر] ، پایان = "")

# چاپ ("Fitnees:" ، fitnessValue ، end = "")

# چاپ ("Eta:" ، etaNumerator)

# اکنون باید فیتنسهمه لبه های متصل به لبه فعلی را پیدا کنیم.

# در این صورت فقط ما می توانیم مقدار eta مخرج را پیدا کنیم

      etaDenominator = 0

# حالا تمام لبه های مجاور را بگیرید

      for n in range(len(legal\_moves)):

# حرکت را به مکعب اعمال کنید

# حرکت اول # حرکتهای مجاور

        properScramble = convert2Scramble(krill\_AllMoves[krill] + legal\_moves[m] + legal\_moves[n] + " ")

# اندازه بهم ریختن مکعب را پیدا کنید

        noofMmoves = len(properScramble)

  # آن بهم ریختن مکعب را انجام دهید و جهت گیری کنید

        scramble, cube\_orientation = Scramble.scramble(True, properScramble, noofMmoves)

      # اکنون مقدار فیتنس جهت گیری مکعب را پیدا کنید

        fitnessValue = fit.fitness(cube\_orientation)

# اکنون مقدار eta در شماره گیر (کیفیت لبه = 1 / Lk یعنی معکوس فاصله) است

        eta = (1/fitnessValue)

# اکنون برای بدست آوردن مقدار مخرج باید مقادیر موجود در واحد ارزش را جمع کنیم

        etaDenominator = etaDenominator + eta

        # چاپ ("\ n")

# چاپ ("انتقال:" ، حرکت\_قانونی [n] ، پایان = "")

# چاپ ("Fitnees:" ، fitnessValue ، end = "")

# چاپ ("اتا:" ، eta ، پایان = "")

# چاپ ("مخرج Eta:" ، etaDenominator)

# حالا احتمال لبه را پیدا کنید

      probability = (etaNumerator/etaDenominator)

# اکنون مقدار احتمال را در فرهنگ لغت احتمال به لبه اختصاص دهید

      probabilityDict[legal\_moves[m]] = probability

# چاپ ("\ n")

# چاپ ("حرکت:" ، حرکت\_قانونی [متر])

# چاپ ("احتمال:" ، احتمال)

# چاپ (احتمال دیکتاتور)

# اکنون فرهنگ لغت احتمال را برای مورچه چاپ کنید

# چاپ (احتمال دیکتاتور)

# این توزیع احتمال به 1 اضافه نمی شود.

# بنابراین ما باید تمام احتمالات را گرفته و آنها را بر مجموع تقسیم کنیم

    sum = 0

    for i in probabilityDict:

      sum = sum + probabilityDict[i]

    # چاپ ("جمع:" ، جمع)

# تقسیم هر توزیع احتمال را بر مجموع

    for i in probabilityDict:

      probabilityDict[i] = probabilityDict[i]/sum

# اکنون فرهنگ لغت احتمال را برای مورچه چاپ کنید

# چاپ ("\ n")

# چاپ (احتمال دیکتاتور)

# حالا مطمئن شوید که مجموع 1 است

#sum = 0

# برای من احتمال

# جمع = جمع + احتمال دیکتاتور [من]

# چاپ ("جمع:" ، جمع)

# اکنون باید فرهنگ لغت احتمال را به ترتیب صعودی مرتب کنیم

# این فرهنگ لغت را مرتب می کند ، اما لیستی از tuple را باز می گرداند [() ، () ، ..]

    dict = sorted(probabilityDict.items(), key=lambda x: x[1])

# بنابراین ما باید لیست tuples را به فرهنگ لغت تبدیل کنیم

    probabilityDict = {}

    for key,value in dict:

      probabilityDict[key] = value

    # اکنون فرهنگ لغت خود را با مقادیر مرتب شده بازگردانده ایم

# چاپ ("\ n")

# چاپ (احتمال دیکتاتور)

# حالا باید جمع تجمعی را پیدا کنیم

# ما برای نگه داشتن حرکت و توزیع تجمعی باید یک فرهنگ لغت ایجاد کنیم

    cumulativeDict = {}

    cumSum = 0

    for key,value in probabilityDict.items():

      cumSum = cumSum + value

      cumulativeDict[key] = cumSum

    # چاپ ("\ n")

# چاپ (قانون تجمعی)

# حالا باید یک عدد تصادفی انتخاب کنیم

    number = random.random()

# چاپ ("\ n")

# چاپ ("شماره تصادفی:" ، شماره)

# حالا باید پیدا کنیم که شماره تصادفی در کجا فرود می آید

    move = ""

    for key, value in cumulativeDict.items():

      # اگر تعداد تولید شده کمتر یا برابر با مقدار باشد ، حرکت مورد نیاز را انجام داده ایم.

# بعد از آن فقط از حلقه خارج شوید.

اگر عدد <= مقدار:

        move = key

        break

    # حرکت و مقدار فیتنس آن حرکت را برگردانید

    return move, fitnessDict[move]

# این تابع تمام گروه میگو هایی را که در یک گره جمع شده اند نشان می دهد

def ShowGroup(nPop, krill\_Orientation):

# در جایی که میگو موجود است چاپ کنید

  group = {}

  for k in range(nPop):

    if krill\_Orientation[k] not in group.keys():

      group[krill\_Orientation[k]] = [k]

    else:

      group[krill\_Orientation[k]].append(k)

  for key,value in group.items():

    print(key, " : ",value)

# این تابع تابع تولید مثل را انجام می دهد

def Reproduction(krill1, krill2, krill\_OrientationMoves, originalScramble, krill\_fitness, krill\_Orientation, krill\_AllMoves, solution, iteration,nPop,start\_time):

  """

krill1 - اولین میگو

میگو 2 - میگو دوم

n: نقطه ژن تصادفی

krill\_OrientationMoves: حرکاتی اعمال می شود که جهت فعلی را بدست آورند

originalScramble: تقارن اولیه ای که روی مکعب اعمال می شود

krill\_fitness: مناسب میگو

krill\_Orientation: جهت گیری krill

krill\_AllMoves: تمام حرکات اعمال شده به مکعب ، از جمله بهم ریختن مکعب

solution: یک ساختار داده برای نگهداری تمام میگوهایی که راه حل را پیدا کرده اند

تکرار: تکرار فعلی

nPop: تعداد کل جمعیت میگو

start\_time: زمانی که حلقه KHO اجرا را شروع می کند  """

# حرکاتی که روی مکعب اعمال می شود به صورت رشته ای است

# رشته را به لیست هر دو میگو والد تبدیل کنید

# آخرین عنصر یک فاصله خواهد بود ، آن را حذف کنید

  krill1Moves = krill\_OrientationMoves[krill1].split(" ")[:-1]

  krill2Moves = krill\_OrientationMoves[krill2].split(" ")[:-1]

# اکنون نقطه ژن تصادفی را پیدا کنید. ما می توانیم از هر یک از لیست های krill1Moves یا krill2Moves استفاده کنیم زیرا اندازه هر دو یکسان است

  n = random.randint(1,len(krill1Moves))

# حالا آنها را تقسیم کنید و داستان را عوض کنید

  childMoveList = krill1Moves[:n] + krill2Moves[n:]

# اکنون این یک لیست خواهد بود. دوباره به رشته تبدیل کنید

  newMove = " ".join([str(elem) for elem in childMoveList])

  print("Parents : ",krill1, "and ",krill2)

  print("Move Sequence of krill : ",krill1," = ",krill\_OrientationMoves[krill1])

  print("Move Sequence of krill : ",krill2," = ",krill\_OrientationMoves[krill2])

  print("Gene Point : ",n)

# اکنون از آنجا که این یک میگو جدید است ، باید آن را به ساختار داده پیوست کنیم

# اکنون حرکتی را که میگو اعمال کرده ذخیره کنید

 krill\_OrientationMoves.append(newMove + " ")

# اکنون حرکت را به مکعب اعمال کنید

  properScramble = convert2Scramble(originalScramble + newMove + " ")

# اندازه بهم ریختن مکعب را پیدا کنید

  noofMmoves = len(properScramble)

# آن بهم ریختن مکعب را انجام دهید و جهت گیری کنید

  scramble, cube\_orientation = Scramble.scramble(True, properScramble, noofMmoves)

  fitnessValue = fit.fitness(cube\_orientation)

  print("New Krill : ", nPop,end=" ")

  print("Move Sequence : ", newMove,end=" ")

  print("Orientation : ",cube\_orientation,end=" ")

  print("Fitnees : ",fitnessValue)

  krill\_fitness.append(fitnessValue)

# اکنون جهت را ذخیره کنید

  krill\_Orientation.append(cube\_orientation)

# اکنون حرکت اعمال شده را ذخیره کنید

  krill\_AllMoves.append(scramble)

# اگر مکعب حل شد

# این به انتهای ساختار داده پیوست می شود

  if krill\_Orientation[-1] == "yyyyyyyyyooooooooogggggggggwwwwwwwwwrrrrrrrrrbbbbbbbbb":

# اندازه راه حل را پیدا کنید

# حرکات اعمال شده یک رشته است. بنابراین آن را به فاصله تقسیم کرده و آخرین کاراکتر را حذف کنید زیرا یک فاصله است.

# این برای یافتن اندازه راه حل لازم است.

# اگر رشته ای بود ، فاصله ها را می شمرد.

    solutionMoves = krill\_OrientationMoves[-1].split(" ")[:-1]

    intermediate\_time = time.time() # این جایی است که یک ذره حالت حل شده را پیدا می کند

solve\_time = intermediate\_time - start\_time

# زمان یافتن راه حل برای یک ذره

# تکرار ، میگو ، جهت گیری ، راه حل ، اندازه راه حل ، زمان

#از آنجا که این میگو جدید متولد شده است ، به انتهای هر ساختار داده اضافه می شود

    solution.append([iteration, nPop, krill\_Orientation[-1], krill\_OrientationMoves[-1], len(solutionMoves), solve\_time])

# این تابع جهش را انجام می دهد

def Mutation(fitnessThreshold, nPop, krill\_fitness, krill\_AllMoves, krill\_Orientation, krill\_OrientationMoves, legal\_moves,originalScramble,start\_time):

  """

  fitnessThreshold: آستانه ای که در آن تصمیم می گیریم که آیا یک میگو مناسب است یا نه

nPop: اندازه جمعیت

krill\_fitness: ساختار داده ای که فیتنسهمه میگو ها را حفظ می کند

krill\_AllMoves: ساختار داده ای است که تمام حرکات اعمال شده به مکعب را با شروع با تقارن در خود نگه می دارد

krill\_Orientation: یک ساختار داده ای است که جهت گیری همه میگو ها را حفظ کند

krill\_OrientationMoves: ساختار داده ای که حرکات اعمال شده به مکعب را پس از تقارن نگه می دارد

legal\_moves: مجموعه تمام حرکات ممکن است که می تواند روی مکعب اعمال شود

originalScramble: تقارن اعمال شده بر روی مکعب

start\_time: زمانی که حلقه اجرای خود را شروع می کند  """

  print("Threshold : ",fitnessThreshold)

  # از هر میگو استفاده کرده و بررسی کنید که آیا فیتنس آن از آستانه بیشتر است

  for krill in range(nPop):

    print("Krill : ",krill,end = " ")

    print("Fitnees : ",krill\_fitness[krill],end= " ")

# اگر فیتنس میگو کمتر از آستانه است ، هیچ جهشی وجود ندارد

    if krill\_fitness[krill] <= fitnessThreshold:

      print("No mutation")

    # دیگر جهش را انجام دهید

    else:

      print("Mutation is needed")

      print("Old Moves : ",krill\_OrientationMoves[krill],end=" ")

      print("Old Orientation : ",krill\_Orientation[krill],end=" ")

# حرکاتی که به میگو اعمال می شود به صورت یک رشته است.

# آن را به یک لیست تبدیل کنید ، عنصر آخر را حذف کنید زیرا یک فاصله است

      krillMoves = krill\_OrientationMoves[krill].split(" ")[:-1]

      # اکنون تصمیم بگیرید که چند نقطه ژنی را که می خواهید تغییر دهید

      points = random.randint(1,len(krillMoves))

      print("Points : ",points)

      # اکنون از یک لیست برای ذخیره تمام شاخص تمام نقاط ژنی که می خواهید تغییر دهید استفاده کنید

      indexList = []

      # این حلقه را اجرا کنید تا نقاط شاخص برابر با نقاطی را که می خواهیم تغییر دهیم پیدا کنیم

      while(len(indexList) != points):

        # یک نقطه شاخص تصادفی ایجاد کنید

# اطمینان حاصل کنید که نقاط شاخص در مقدار مناسب قرار دارند

# از اندازه فیلم 7 است ، سپس شاخص 0 تا 6 است

        n = random.randint(0,len(krillMoves)-1)

# اگر این نقطه شاخص در لیست نیست

        if n not in indexList:

          indexList.append(n)

# اگر نقطه تولید شده در لیست موجود است ، ادامه دهید

        else:

          continue

# حالا هر شاخصی را که پیدا کردیم بگیرید

      for i in indexList:

      # حرکت را در آن شاخص با یک حرکت تصادفی تغییر دهید

        krillMoves[i] = random.choice(legal\_moves)

# اکنون این یک لیست خواهد بود. دوباره به رشته تبدیل کنید

      newMove = " ".join([str(elem) for elem in krillMoves])

# اکنون از آنجا که حرکات اعمال شده بر میگو تغییر کرده است ، حرکت را در ساختار داده تغییر دهید

      krill\_OrientationMoves[krill] = newMove + " "

     # اکنون حرکت را به مکعب اعمال کنید

      properScramble = convert2Scramble(originalScramble + newMove + " ")

     # اندازه بهم ریختن مکعب را پیدا کنید

      noofMmoves = len(properScramble)

     # آن بهم ریختن مکعب را انجام دهید و جهت گیری کنید

      scramble, cube\_orientation = Scramble.scramble(True, properScramble, noofMmoves)

      fitnessValue = fit.fitness(cube\_orientation)

      print("New Moves : ", newMove,end=" ")

      print("New Orientation : ",cube\_orientation,end=" ")

      print("New Fitnees : ",fitnessValue)

      krill\_fitness[krill] = fitnessValue

      # اکنون جهت را ذخیره کنید

      krill\_Orientation[krill] = cube\_orientation

      # اکنون حرکت اعمال شده را ذخیره کنید

      krill\_AllMoves[krill] = scramble

# اگر مکعب حل شد

# این به انتهای ساختار داده پیوست می شود

      if krill\_Orientation[krill] == "yyyyyyyyyooooooooogggggggggwwwwwwwwwrrrrrrrrrbbbbbbbbb":

        # اندازه راه حل را پیدا کنید

# حرکات اعمال شده یک رشته است. بنابراین آن را به فاصله تقسیم کرده و آخرین کاراکتر را حذف کنید زیرا یک فاصله است.

# این برای یافتن اندازه راه حل لازم است.

# اگر رشته ای بود ، فاصله ها را می شمرد.

        solutionMoves = krill\_OrientationMoves[krill].split(" ")[:-1]

        intermediate\_time = time.time() #این جایی است که یک ذره حالت حل شده را پیدا می کند

solve\_time = intermediate\_time - start\_time

# زمان یافتن راه حل برای یک ذره

# تکرار ، میگو ، جهت گیری ، راه حل ، اندازه راه حل ، زمان

# از آنجا که این میگو جدید متولد شده است ، به انتهای هر ساختار داده اضافه می شود        solution.append([iteration, krill, krill\_Orientation[krill], krill\_OrientationMoves[krill], len(solutionMoves), solve\_time])

# تعریف تابع

def KHO(iterations, krills, flag, scramble):

# تعریف مشکل برای KHO

# این تقابل و جهت گیری برچسب ها را در مکعب برمی گرداند

# اگر این یک بهم ریختن مکعب تصادفی است

  if flag == False:

   # تعریف مسئله

    scramble, cube\_orientation = Scramble.scramble(flag,scramble,0# این تقابل و جهت گیری برچسب ها را در مکعب برمی گرداند

# دیگر اگر این یک بهم ریختن مکعب دستی باشد

  else:

# اکنون حرکت را به مکعب اعمال کنید

    properScramble = convert2Scramble(scramble)

    # اندازه بهم ریختن مکعب را پیدا کنید

    noofMmoves = len(properScramble)

    # آن بهم ریختن مکعب را انجام دهید و جهت گیری کنید

    scramble, cube\_orientation = Scramble.scramble(flag, properScramble, noofMmoves)

  # بهم ریختن مکعب استفاده شده را ذخیره کنید ، شاید بخواهیم بعداً آن را نشان دهیم

  originalScramble = scramble

  # جهت گیری مکعب اولیه را ذخیره کنید ، شاید بخواهیم بعداً آن را نشان دهیم

  initialOrientation = cube\_orientation

  # نمره اصلی Fitnees ، شاید بخواهیم بعداً آن را نشان دهیم

  originalFitness = fit.fitness(cube\_orientation)

  # جهت گیری اولیه را چاپ کنید

  print("Scramble : ",originalScramble)

  print("Initial cube orientation: ", cube\_orientation)

  print("Initial Fitnees : ",originalFitness)

  print("\n")

  # پارامترهای بهینه سازی کلنی میگو

  maxIter = iterations  # حداکثر تعداد تکرار

  nPop = originalPopulation = krills  # حداکثر اندازه جمعیت یعنی تعداد کل میگو ها

  # حداکثر تکرار و کل جمعیت میگو ها را چاپ کنید

  print("Maximum Iteration : ",maxIter)

  print("Total Population : ",nPop)

  # اعلامیه متغیر ما باید ساختارهای داده ای ایجاد کنیم تا برخی از مقادیر قابل تغییر را در خود نگه داریم

  # یک ساختار داده برای نگه داشتن حرکات قانونی در دسترس یعنی U ، U '، U2 ، ... ، R ، R' ، R2.

# از آنجا که Kociemba روی متریک نیمه دور کار می کند ، جایی که دو دور ضلع به عنوان یک حرکت محاسبه می شود ، ما از معیار نیمه دور استفاده خواهیم کرد.

  legal\_moves = ["U", "U'", "U2", "D", "D'","D2", "L","L'", "L2", "R", "R'", "R2", "F","F'", "F2", "B", "B'", "B2"]

# این ساختار داده ، تمام حرکات اعمال شده از حالت اولیه را که از تقارن شروع می شود ، در خود نگه می دارد

  krill\_AllMoves = []

# یک ساختار داده برای نگه داشتن موقعیت هر میگو یعنی اینکه در کدام جهت قرار دارند

  krill\_Orientation = []

# یک ساختار داده برای نگه داشتن حرکاتی که باعث شده است میگو به موقعیت فعلی برسد ، شامل بهم ریختن مکعب نیست

  krill\_OrientationMoves = []

  # این متغیر مقدار فیتنس هر میگو را در خود نگه می دارد

  krill\_fitness = []

# از این لیست برای نگهداری تمام میگو هایی که حالت حل شده را پیدا کرده اند و سایر اطلاعات مانند تکرار آن برای یافتن راه حل ، اندازه و غیره استفاده می شود.

  solution = []

# مقداردهی اولیه متغیرها. در اینجا ما می خواهیم تمام میگو ها را مقدار دهی کنیم

  for i in range(nPop):

    # ما قصد داریم تقارن استفاده شده را ضمیمه کنیم زیرا این تنها حرکتی بود که تاکنون انجام شده است

    krill\_AllMoves.append(scramble)

# در ابتدا تمام میگو ها در موقعیت اولیه درهم قرار می گیرند

    krill\_Orientation.append(cube\_orientation)

# در ابتدا از آنجا که هیچ حرکتی انجام نشده است ، ساختار داده خالی است

    krill\_OrientationMoves.append("")

    # مقدار فیتنس برای همه میگو ها ، مقدار فیتنس برای حالت مخلوط شده خواهد بود

    krill\_fitness.append(originalFitness)

  """

# اکنون جزئیات مربوط به هر میگو را چاپ کنید

  for i in range(nPop):

    print("Krill : ",i)

    print("All moves : ",krill\_AllMoves[i])

    print("Orientation : ",krill\_Orientation[i])

    print("Moves applied to reach orientation from scrambled state : ",krill\_OrientationMoves[i])

    print("Fitnees : ",krill\_fitness[i])

    print("\n")

  """

# از این متغیر برای ترسیم نمودار استفاده می شود

  xIter = []

  yFitnessCurrent = []

  yFitnessGlobal = []

# تعداد تکرارها

  start\_time = time.time()  # این همان جایی است که اعدام آغاز می شود

  # ادامه برای تعداد کل تکرارها

  for iteration in range(maxIter):

    # برای رسم باید مقادیری پیدا کنیم

    xIter.append(iteration)

    sumFitness = 0

    for i in range(len(krill\_fitness)):

      sumFitness = sumFitness + krill\_fitness[i]

    yFitnessCurrent.append(sumFitness/nPop) # ما می خواهیم میانگین فیتنس همه میگوها را برای تکرار ضمیمه کنیم

    yFitnessGlobal.append(min(krill\_fitness))

    # اگر همه میگو ها راه حل پیدا کرده اند ، از حلقه خارج شوید

    if len(solution) == nPop:

      break

    #system('cls')

    print("\n")

    print("Iteration : ",iteration)

    ##\*\*\*\*\*\*\*\*\*\* انتخاب یال \*\*\*\*\*\*\*\*\*\*##

    print("\*\*\*\*\*\*\*\*\*\* Edge Selection Stage \*\*\*\*\*\*\*\*\*\*")

    # برای همه میگو های موجود در جمعیت

    for krill in range(nPop):

      # چاپ کنید که کدام میگو است

      print("Krill : ",krill,end = " ")

      # جهت میگو را چاپ کنید

      print("Orientation : ",krill\_Orientation[krill],end = " ")

      # فیتنس فعلی را به صورت میگو چاپ کنید

      print("Current Fitnees : ",krill\_fitness[krill],end= " ")

      if krill\_Orientation[krill] == "yyyyyyyyyooooooooogggggggggwwwwwwwwwrrrrrrrrrbbbbbbbbb":

        print("Solution found for krill")

      # اگر میگو به حالت حل نشده رسیده باشد.

      if krill\_Orientation[krill] != "yyyyyyyyyooooooooogggggggggwwwwwwwwwrrrrrrrrrbbbbbbbbb":

        # اکنون egde / move بعدی را برای میگو انتخاب کرده و فیتنس آن را پیدا کنید        move, fitnessValue1 = ReturnMove(legal\_moves,krill\_AllMoves,krill\_Orientation,krill\_fitness,krill)

        # حرکت اعمال شده را چاپ کنید

        print("Move : ",move,end=" ")

        # اکنون حرکتی را که میگو اعمال کرده ذخیره کنید

        krill\_OrientationMoves[krill] = krill\_OrientationMoves[krill] + move + " "

        # اکنون حرکت را به مکعب اعمال کنید

        properScramble = convert2Scramble(krill\_AllMoves[krill] + move + " ")

        # اندازه بهم ریختن مکعب را پیدا کنید

        noofMmoves = len(properScramble)

        # آن بهم ریختن مکعب را انجام دهید و جهت گیری کنید

        scramble, cube\_orientation = Scramble.scramble(True, properScramble, noofMmoves)

# حالا مقدار فیتنس را پیدا کنید و آن را ذخیره کنید

# ما می دانیم که برخی از ایالات ، ظرافت های چندگانه ای دارند.

# وقتی دولت را چندین بار به Kociemba می دهیم ، گاهی اوقات فیتنس متفاوتی خواهد داشت

# بنابراین در اینجا با دو بار اجرای Kociemba در همان حالت ، حداقل مقدار نازک بودن را پیدا می کنیم

        fitnessValue2 = fit.fitness(cube\_orientation)

        fitnessValue = min([fitnessValue1, fitnessValue2])

        print("New Fitnees : ",fitnessValue)

        krill\_fitness[krill] = fitnessValue

        # اکنون جهت را ذخیره کنید

        krill\_Orientation[krill] = cube\_orientation

        # اکنون حرکت اعمال شده را ذخیره کنید

        krill\_AllMoves[krill] = scramble

        # اگر مکعب حل شود

        if krill\_Orientation[krill] == "yyyyyyyyyooooooooogggggggggwwwwwwwwwrrrrrrrrrbbbbbbbbb":

# اندازه راه حل را پیدا کنید

# حرکات اعمال شده یک رشته است. بنابراین آن را به فاصله تقسیم کرده و آخرین کاراکتر را حذف کنید زیرا یک فاصله است.

# این برای یافتن اندازه راه حل لازم است.

# اگر رشته ای بود ، فاصله ها را می شمرد.

          solutionMoves = krill\_OrientationMoves[krill].split(" ")[:-1]

          intermediate\_time = time.time()

# این جایی است که یک ذره حالت حل شده را پیدا می کند

 solve\_time = intermediate\_time - start\_time

# زمان یافتن راه حل برای یک ذره

# تکرار ، میگو ، جهت گیری ، راه حل ، اندازه راه حل ، زمان

          solution.append([iteration, krill, krill\_Orientation[krill], krill\_OrientationMoves[krill], len(solutionMoves),solve\_time])

    #  گروه میگو ها را که در یک گره جمع شده اند نشان دهید

    ShowGroup(nPop, krill\_Orientation)

    ## \*\*\*\*\*\*\*\*\*\* کشتن میگو که مناسب نیست \*\*\*\*\*\*\*\*\*\* ##

# ما میگوی را که دارای فیتنس بیشتر از 50٪ از حداقل مقدار جهانی باشد ، از بین خواهیم برد

# اکنون مقدار حداقل جهانی را پیدا کنید

    globalMin = min(krill\_fitness)

# آستانه فیتنس را تنظیم کنید

    fitnessThreshold = (globalMin // 2) + globalMin

# از این متغیر برای پیگیری همه میگو هایی که کشته شده اند استفاده می شود

    killed = 0

    print("\n\n")

    print("\*\*\*\*\*\*\*\*\*\* Killing Stage \*\*\*\*\*\*\*\*\*\*")

    print("Global Minimum : ",globalMin)

    print("Fitnees Threshold : ", fitnessThreshold)

# هر میگو مصرف کنید

# عنصر پاپ از پشت

    for krill in range(nPop-1,-1,-1):

      # اگر فیتنس بیش از آستانه است ، آن را از بین ببرید

      if krill\_fitness[krill] > fitnessThreshold:

# تمام پارامترهای مرتبط با میگو را حذف کنید

# همچنین از آنجا که ما از pop استفاده می کنیم ، عنصر بعدی به شاخص فعلی هدایت می شود

        print("Killed : ",krill)

        krill\_AllMoves.pop(krill)

        krill\_Orientation.pop(krill)

        krill\_OrientationMoves.pop(krill)

        krill\_fitness.pop(krill)

        # تعداد میگوهای کشته شده را افزایش دهید

        killed += 1

    # اندازه جمعیت را کاهش دهید

    nPop = nPop - killed

    if killed == 0:

      print("No krill was killed")

    # اکنون گروه را نشان دهید:

    ShowGroup(nPop, krill\_Orientation)

## \*\*\*\*\*\*\*\*\*\* در اینجا ما می خواهیم تولید مثل انجام دهیم تا اطمینان حاصل کنیم که جمعیت به حالت عادی برگشته است \*\*\*\*\*\*\*\*\*\* ##

# ما می خواهیم تولید مثل را انجام دهیم تا زمانی که جمعیت به حالت عادی برگردد

    print("\n\n")

    print("\*\*\*\*\*\*\*\*\*\* Reproduction Stage \*\*\*\*\*\*\*\*\*\*")

# اگر هیچ میگوی کشته نشد ، دیگر نیازی به تولید مثل نیست

    if killed == 0:

      print("No reproduction as no krill was killed.")

    # اگر فقط یک میگو باقی مانده باشد ، پس ما نمی توانیم تولید مثل کنیم

    elif nPop == 1:

        print("No reproduction possible as only one krill is left.")

    else:

      print("Reproduction")

      populationRestored = False

      while nPop != originalPopulation:

        # حرکات اعمال شده به اولین میگو را انجام دهید

        for i in range(len(krill\_OrientationMoves)):

          # میگو بعدی را انتخاب کنید

          for j in range(len(krill\_OrientationMoves)):

            # اگر i و j یکسان هستند ، به میگو بعدی بروید ، زیرا ما نمی خواهیم میگو با خودش تولید مثل کند

            if i == j:

              continue

            # تولید مثل بین میگو 1 و میگو 2 و افزایش اندازه جمعیت

            print("\n")

            print("Reproduction between krill ",i, " and krill ",j)

            Reproduction(i, j, krill\_OrientationMoves, originalScramble, krill\_fitness, krill\_Orientation, krill\_AllMoves, solution, iteration,nPop, start\_time)

            nPop += 1

            # در صورت بازیابی جمعیت ، از حلقه خارج شوید

            if nPop == originalPopulation:

              print("Population restored to ",nPop)

              populationRestored = True

              break

            # سایر موارد تولید مثل بین میگو 2 و میگو 1 را انجام می دهند و اندازه جمعیت را افزایش می دهند

            print("\n")

            print("Reproduction between krill ",j, " and krill ",i)

            Reproduction(j, i, krill\_OrientationMoves, originalScramble, krill\_fitness, krill\_Orientation, krill\_AllMoves, solution, iteration,nPop,start\_time)

            nPop += 1

            # در صورت بازیابی جمعیت ، از حلقه خارج شوید

            if nPop == originalPopulation:

              print("Population restored to ",nPop)

              populationRestored = True

              break

          # شکستن حلقه for i

          if populationRestored:

            break

        # از حلقه while بیرون بیایید

        if populationRestored:

          break

## \*\*\*\*\*\*\*\*\*\* در اینجا ما می خواهیم جهش انجام دهیم ، به عنوان مثال تغییر توالی حرکت میگو که دارای فیتنس بیش از آستانه است \*\*\*\*\*\*\*\*\*\* ##

# جهش میگو ، که فیتنس آن بیشتر از آستانه است

# این ممکن است زمانی اتفاق بیفتد که کودک آمادگی جسمانی بیش از آستانه داشته باشد

    print("\n\n")

    print("\*\*\*\*\*\*\*\*\*\* Mutation Stage \*\*\*\*\*\*\*\*\*\*")

    # تابع جهش را فراخوانی کنید

    Mutation(fitnessThreshold, nPop, krill\_fitness, krill\_AllMoves, krill\_Orientation, krill\_OrientationMoves,legal\_moves, originalScramble,start\_time)

  end\_time = time.time()  # این جایی است که کل حلقه PSO اجرا را به پایان می رساند

  total\_time = end\_time - start\_time  # این کل زمان مورد نیاز برای اجرای حلقه است

  # جهت گیری مکعب اولیه را چاپ کنید

  print("\n")

  print("Cube orientation: Yellow on top and green on front.")

  print("Reading order goes: Up, Right, Front, Down, Left, Back.")

  print("Scramble: ", originalScramble)

  print("Initial cube orientation : ",initialOrientation)

  # جزئیات راه حل را برای هر میگو چاپ کنید

  for sol in solution:

    print("\n")

    print("Iteration : ",sol[0])

    print("Krill : ",sol[1])

    print("Orientation : ", sol[2])

    print("Solution : ",sol[3])

    print("Length : ",sol[4])

    print("Time : ",sol[5])

  # اگر راه حل پیدا نشد ، اطلاعات مناسب را چاپ کنید

  if len(solution)==0:

    print("\n")

    print("Solution was not found by any krill")

    print("The total number of krill that found solution: ",len(solution),"/",originalPopulation) # 0 ذره آن را حل کرد

    print("Original Solution Length: ", originalFitness)

    print("Average solution length: NIL") # از آنجا که راه حل یافت نشد

    print("Iteration at which the first solution was found: NIL") # از آنجا که راه حل یافت نشد

    print("Iteration at which the last krill found the solution: NIL")  # از آنجا که راه حل یافت نشد

    print("Average iterations it takes for the next krill to reach solved state: NIL")  # از آنجا که راه حل یافت نشد

    print("Total time for loop execution : ",total\_time)

# در غیر انصورت اگر راه حل پیدا شد

  else:

    # چاپ جزئیات:

# اندازه متوسط راه حل را پیدا کنید

    print("\n")

    sumSol = 0

    for sol in solution:

      sumSol = sumSol + sol[4]

    averageSol = sumSol/len(solution)

    print("The total number of krill that found solution: ",len(solution),"/",originalPopulation)

    print("Original Solution Length: ", originalFitness)

    print("Average solution length: ", averageSol)

    # میانگین تکرار و زمان لازم برای رسیدن ذره بعدی به حالت حل شده را پیدا کنید

    sumIter = 0

    sumIterTime = 0

    for i in range(len(solution)):

      if (i+1) >= len(solution):

        break

      else:

        sumIter = sumIter + (solution[i+1][0]-solution[i][0])

        sumIterTime = sumIterTime + (solution[i+1][5]-solution[i][5])

    averageIter = sumIter/len(solution)

    averageIterTime = sumIterTime/len(solution)

    # حداقل تکرار که در آن راه حل پیدا شده و همچنین حداکثر تکرار

    print("Iteration at which the first solution was found: ",solution[0][0])

    print("Iteration at which the last krill found the solution: ", solution[-1][0])

    print("Average iterations it takes for the next krill to reach solved state: ",averageIter)

    print("\n")

    print("Time required to find the first solution : ",solution[0][5],"seconds") # در پنجمین شاخص بردار راه حل زمان ذخیره می شود

    print("Total time for KHO loop execution: ", solution[-1][5],"seconds")

    print("Average time it takes for the next krill to reach solved state: ",averageIterTime, "seconds")

  # رسم تغییر بهترین حالت شخصی با هر تکرار برای ذره

  plt.grid(True)

  plt.plot(xIter,yFitnessGlobal,c="blue",linestyle="--",marker="o")

  plt.plot(xIter,yFitnessCurrent,c="red",linestyle=':') # تکرار طرح در مقابل فیتنس

  plt.xlabel('Iterations')  # نامگذاری محور x به عنوان تکرار

  plt.ylabel('Fitness Score') # نام محور y به عنوان نمره فیتنس

  plt.title('Iterations vs Fitness')  # دادن عنوان به نمودار من

  plt.legend(["Global Fitness","Current Fitnees Average"])

  plt.show()

# برنامه اصلی

if \_\_name\_\_ == "\_\_main\_\_":

  print("\n")

  print("Krill Herd Optimization")

  iterations = int(input("Enter the number of iterations : "))

  krills = int(input("Enter the number of krills : "))

  randomOrManual = input("Type M for manual scramble or R for random scrabmle : ")

  if randomOrManual == "R":

    flag = False

    scramble = ""

  elif randomOrManual == "M":

    flag = True

    scramble = input("Type scramble : ")

# تابع تابع

# دلیل ایجاد را به تقابل وارد شده توسط کاربر اضافه کنید.

  KHO(iterations, krills, flag, scramble+" ") # فراخوانی الگوریتم میگو

**FitnessFunction.py**: محاسبه برازندگی

import solver

def fitness(cube\_orientation):

"""

این تابع نمره برازندگی جهت گیری مکعب را ارزیابی می کند.

تابع برازندگی فاصله سه بعدی منهتن نامیده می شود.

در زمینه مکعب روبیک حداقل تعداد حرکات لازم برای حل مکعب از موقعیت فعلی است.

این توسط حل کننده دو فاز Kociemba ارزیابی می شود. """

# ما باید جهت مکعب را تغییر دهیم "UUUUUUUUURRRRRRRRRFFFFFFFFFDDDDDDDDDLLLLLLLLLBBBBBBBBB"

# توالی جهت گیری مکعب ما به شرح زیر است:

# yyyyyyyyyooooooooogggggggggwwwwwwwwwrrrrrrrrrbbbbbbbbb

# توالی جهت گیری مکعب ما به شرح زیر است:

# [y]ellow = [U]P, [o]range = [R]ight, [g]reen = [F]ront, [w]hite = [D]own, [r]ed = [L]eft, [b]lue = [B]ack

cubestring = ""

for i in range(len(cube\_orientation)):

# In the cube orientation if the color is yellow change it to U as yellow is on top

if cube\_orientation[i] == "y":

cubestring = cubestring + "U"

# In the cube orientation if the color is blue change it to B as blue is on back

elif cube\_orientation[i] == "b":

cubestring = cubestring + "B"

# In the cube orientation if the color is red change it to L as red is on left

elif cube\_orientation[i] == "r":

cubestring = cubestring + "L"

# In the cube orientation if the color is green change it to F as red is on front

elif cube\_orientation[i] == "g":

cubestring = cubestring + "F"

# In the cube orientation if the color is orange change it to R as orange is on right

elif cube\_orientation[i] == "o":

cubestring = cubestring + "R"

# In the cube orientation if the color is white change it to D as red is on down

elif cube\_orientation[i] == "w":

cubestring = cubestring + "D"

# آخرین کاراکتر یک راه حل را حذف کنید زیرا فقط یک فاصله است و آن را به فاصله تقسیم کنید تا بتوانیم حرکات را بشماریم

solution = solver.solve(cubestring)[:-1].split(" ")

# Return the length of the solution

return len(solution)

**Scramble.py**: بهم ریختن مکعب

import scrambleGenerator as sg

def scramble(flag, properScramble, noofMoves):

"""

این تابع مکعب را به صورت تصادفی مخلوط می کند و رنگ های هر مکعب را برمی گرداند """

# این قسمت به معنای فقط مخلوط کردن مکعب است

if flag == False:

s, cube = sg.generate()

# این قسمت به معنای اعمال حرکات روی مکعب است

else:

cube = sg.makeMove(properScramble, noofMoves)

s = properScramble

# تابع برازندگی فقط در صورتی برازندگی مکعب را ارزیابی می کند که رنگهای موجود در هر برچسب به آن داده شده باشد.

# بنابراین ما هر یک از برچسب ها را به یک رنگ ترسیم می کنیم

cubelets = [

cube[0][0], cube[0][1], cube[0][2], cube[0][7], 'y', cube[0][3], cube[0][6], cube[0][5], cube[0][4],

cube[3][0], cube[3][1], cube[3][2], cube[3][7], 'o', cube[3][3], cube[3][6], cube[3][5], cube[3][4],

cube[2][0], cube[2][1], cube[2][2], cube[2][7], 'g', cube[2][3], cube[2][6], cube[2][5], cube[2][4],

cube[5][0], cube[5][1], cube[5][2], cube[5][7], 'w', cube[5][3], cube[5][6], cube[5][5], cube[5][4],

cube[1][0], cube[1][1], cube[1][2], cube[1][7], 'r', cube[1][3], cube[1][6], cube[1][5], cube[1][4],

cube[4][0], cube[4][1], cube[4][2], cube[4][7], 'b', cube[4][3], cube[4][6], cube[4][5], cube[4][4]

]

# ما لیست را به رشته تبدیل خواهیم کرد

cube\_orientation = ''.join([str(elem) for elem in cubelets])

scramble = ""

for array in s:

for elem in array:

scramble = scramble + elem

scramble = scramble + " "

return scramble, cube\_orientation

**scrambleGenerator.py:** تولید بهم ریختگی مکعب

|  |
| --- |
|  |

import random

from scrambleImage import scramble

moves = ["U", "D", "F", "B", "R", "L"]

dir = ["", "'", "2"]

slen = random.randint(5,30) # Length of the scramble

def scramble\_gen():

scramble = [0] \* slen

for x in range(len(scramble)):

scramble[x] = [0] \* 2

return scramble

def scramble\_replace(ar):

for x in range(len(ar)):

ar[x][0] = random.choice(moves)

ar[x][1] = random.choice(dir)

return ar

def valid(ar):

for x in range(1, len(ar)):

while ar[x][0] == ar[x-1][0]:

ar[x][0] = random.choice(moves)

for x in range(2, len(ar)):

while ar[x][0] == ar[x-2][0] or ar[x][0] == ar[x-1][0]:

ar[x][0] = random.choice(moves)

return ar

def sprint(ar):

for x in range(len(ar)):

print(str(ar[x][0]) + str(ar[x][1]), end = " ")

# این تابع برای ایجاد حرکت روی مکعب استفاده می شود

def makeMove(s, len):

cube = scramble(s, len)

return cube

# اینجاست که اجرای برنامه یعنی تولید بهم ریختگی مکعب آغاز می شود

def generate():

s = scramble\_replace(scramble\_gen())

valid(s)

sprint(valid(s))

print("[" + str(slen) + "]\n")

cube = makeMove(s, slen)

return s, cube

**solver.py**

|  |
| --- |
|  |

# #######کلاس SolverThread الگوریتم دو فاز را پیاده سازی می کند###############

import face

import threading as thr

import cubie

import symmetries as sy

import coord

import enums as en

import moves as mv

import pruning as pr

import time

class SolverThread(thr.Thread):

def \_\_init\_\_(self, cb\_cube, rot, inv, ret\_length, timeout, start\_time, solutions, terminated, shortest\_length):

"""

: param cb\_cube مکعبی که باید در نمایش CubieCube حل شود

: param rotمکعب را 120 درجه می چرخاند \* در امتداد مورب طولانی قبل از استفاده از الگوریتم دو فاز ، می پوسد

: param inv: 0: مکعب را معکوس نکنید. 1: قبل از استفاده از الگوریتم دو فاز ، مکعب را معکوس کنید

: param ret\_length: اگر با طول <= ret\_length پیدا شود جستجو متوقف می شود.

کارآمدترین راه حل مکعب این است که شش رشته به طور موازی با چرخش = 0 ، 1 و 2 و

inv = 0، 1. اولین موضوعی که راه حل پیدا می کند پرچم خاتمه یافته را تنظیم می کند که تمام رشته های دیگر را علامت می دهد

کوچک کردن به طور متوسط ​​این حل یک مکعب حدود 12 برابر سریعتر از حل یک مکعب با یک نخ است.

و این با وجود Pythons GlobalInterpreterLock GIL.

: param timeout: اساساً حداکثر زمان جستجو در چند ثانیه. اساساً به این دلیل که جستجو بر نمی گردد

قبل از اینکه حداقل یک راه حل پیدا شود

: param start\_time: زمان شروع جستجو.

: param Solutions: آرایه ای با راه حل های یافت شده توسط شش رشته موازی

: param termined: اتفاقی که شش رشته برای نشان دادن درخواست خاتمه به اشتراک گذاشته اند

: param shortest\_length: طول راه حل های کوتاه در آرایه راه حل

"""

thr.Thread.\_\_init\_\_(self)

self.cb\_cube = cb\_cube # مکعب

self.co\_cube = None

#مکعب در اجرای تابع مقداردهی اولیه شد

self.rot = rot

self.inv = inv

self.sofar\_phase1 = None

self.sofar\_phase2 = None

self.lock = thr.Lock()

self.ret\_length = ret\_length

self.timeout = timeout

self.start\_time = start\_time

self.cornersave = 0

# این متغیرها توسط شش رشته به اشتراک گذاشته می شوند که در حل تابع شروع می شوند

self.solutions = solutions

self.terminated = terminated

self.shortest\_length = shortest\_length

def search\_phase2(self, corners, ud\_edges, slice\_sorted, dist, togo\_phase2):

# ##############################################################################################################

if self.terminated.is\_set():

return

################################################################################################################

if togo\_phase2 == 0 and slice\_sorted == 0:

self.lock.acquire() # فاز 2 حل می شود،راه حل ذخیره می شود

man = self.sofar\_phase1 + self.sofar\_phase2

if len(self.solutions) == 0 or (len(self.solutions[-1]) > len(man)):

if self.inv == 1: # ما مکعب معکوس را حل کردیم

man = list(reversed(man))

man[:] = [en.Move((m // 3) \* 3 + (2 - m % 3)) for m in man] # R1->R3, R2->R2, R3->R1 etc.

man[:] = [en.Move(sy.conj\_move[m, 16 \* self.rot]) for m in man]

self.solutions.append(man)

self.shortest\_length[0] = len(man)

if self.shortest\_length[0] <= self.ret\_length: # ما به طول هدف رسیده ایم

self.terminated.set()

self.lock.release()

else:

for m in en.Move:

if m in [en.Move.R1, en.Move.R3, en.Move.F1, en.Move.F3,

en.Move.L1, en.Move.L3, en.Move.B1, en.Move.B3]:

continue

if len(self.sofar\_phase2) > 0:

diff = self.sofar\_phase2[-1] // 3 - m // 3

if diff in [0, 3]: # حرکت پی در پی: در یک صورت یا در یک محور با نظم اشتباه

continue

else:

if len(self.sofar\_phase1) > 0:

diff = self.sofar\_phase1[-1] // 3 - m // 3

if diff in [0, 3]: # حرکت پی در پی: در یک صورت یا در یک محور با نظم اشتباه

continue

corners\_new = mv.corners\_move[18 \* corners + m]

ud\_edges\_new = mv.ud\_edges\_move[18 \* ud\_edges + m]

slice\_sorted\_new = mv.slice\_sorted\_move[18 \* slice\_sorted + m]

classidx = sy.corner\_classidx[corners\_new]

sym = sy.corner\_sym[corners\_new]

dist\_new\_mod3 = pr.get\_corners\_ud\_edges\_depth3(

40320 \* classidx + sy.ud\_edges\_conj[(ud\_edges\_new << 4) + sym])

dist\_new = pr.distance[3 \* dist + dist\_new\_mod3]

if max(dist\_new, pr.cornslice\_depth[24 \* corners\_new + slice\_sorted\_new]) >= togo\_phase2:

continue # دسترسی به مکعب حل شده غیر ممکن است togo\_phase2 - 1 moves

self.sofar\_phase2.append(m)

self.search\_phase2(corners\_new, ud\_edges\_new, slice\_sorted\_new, dist\_new, togo\_phase2 - 1)

self.sofar\_phase2.pop(-1)

def search(self, flip, twist, slice\_sorted, dist, togo\_phase1):

# ##############################################################################################################

if self.terminated.is\_set():

return

################################################################################################################

if togo\_phase1 == 0: # فاز 1 حل شده

if time.monotonic() > self.start\_time + self.timeout and len(self.solutions) > 0:

self.terminated.set()

# compute initial phase 2 coordinates

if self.sofar\_phase1: # چک کن اگر لیست خالی نیست

m = self.sofar\_phase1[-1]

else:

m = en.Move.U1

# مقدار در اینجا بی ربط است ، هیچ فاز 1 حرکت نمی کند

if m in [en.Move.R3, en.Move.F3, en.Move.L3, en.Move.B3]: # فاز 1 راه حل به صورت جفت است

corners = mv.corners\_move[18 \* self.cornersave + m - 1] # apply R2, F2, L2 ord B2 on last ph1 solution

else:

corners = self.co\_cube.corners

for m in self.sofar\_phase1: # پیکربندی کنونی گوشه را دریافت کنید

corners = mv.corners\_move[18 \* corners + m]

self.cornersave = corners

# راه حل جدید باید کوتاهتر باشد و ما از فاز 2 استفاده نمی کنیم

maneuvers with length > 11 - 1 = 10

togo2\_limit = min(self.shortest\_length[0] - len(self.sofar\_phase1), 11)

if pr.cornslice\_depth[24 \* corners + slice\_sorted] >= togo2\_limit: # این پیش بررسی بازگشت محاسبات را تسریع می کند

u\_edges = self.co\_cube.u\_edges

d\_edges = self.co\_cube.d\_edges

for m in self.sofar\_phase1:

u\_edges = mv.u\_edges\_move[18 \* u\_edges + m]

d\_edges = mv.d\_edges\_move[18 \* d\_edges + m]

ud\_edges = coord.u\_edges\_plus\_d\_edges\_to\_ud\_edges[24 \* u\_edges + d\_edges % 24]

dist2 = self.co\_cube.get\_depth\_phase2(corners, ud\_edges)

for togo2 in range(dist2, togo2\_limit): # بیش از togo2\_limit استفاده نکنید - 1 حرکت در مرحله 2 است

self.sofar\_phase2 = []

self.search\_phase2(corners, ud\_edges, slice\_sorted, dist2, togo2)

else:

for m in en.Move:

dist = 0 به این معنی است که ما در زیر گروه H هستیم. اگر کمتر از 5 حرکت باقی مانده باشد

# این باعث می شود تمام حرکات باقی مانده حرکات فاز 2 باشد. بنابراین می توانیم این موارد را در پایان فاز 1 منع کنیم

# و این حرکات را در فاز 2 انجام دهیم

if dist == 0 and togo\_phase1 < 5 and m in [en.Move.U1, en.Move.U2, en.Move.U3, en.Move.R2,

en.Move.F2, en.Move.D1, en.Move.D2, en.Move.D3,

en.Move.L2, en.Move.B2]:

continue

if len(self.sofar\_phase1) > 0:

diff = self.sofar\_phase1[-1] // 3 - m // 3

if diff in [0, 3]: # حرکتهای پی در پی: در یک صورت یا در همان محور با نظم اشتباه

continue

flip\_new = mv.flip\_move[18 \* flip + m] # N\_MOVE = 18

twist\_new = mv.twist\_move[18 \* twist + m]

slice\_sorted\_new = mv.slice\_sorted\_move[18 \* slice\_sorted + m]

flipslice = 2048 \* (slice\_sorted\_new // 24) + flip\_new # N\_FLIP \* (slice\_sorted // N\_PERM\_4) + flip

classidx = sy.flipslice\_classidx[flipslice]

sym = sy.flipslice\_sym[flipslice]

dist\_new\_mod3 = pr.get\_flipslice\_twist\_depth3(2187 \* classidx + sy.twist\_conj[(twist\_new << 4) + sym])

dist\_new = pr.distance[3 \* dist + dist\_new\_mod3]

if dist\_new >= togo\_phase1: # impossible to reach subgroup H in togo\_phase1 - 1 moves

continue

self.sofar\_phase1.append(m)

self.search(flip\_new, twist\_new, slice\_sorted\_new, dist\_new, togo\_phase1 - 1)

self.sofar\_phase1.pop(-1)

def run(self):

cb = None

if self.rot == 0: #بدون چرخش

cb = cubie.CubieCube(self.cb\_cube.cp, self.cb\_cube.co, self.cb\_cube.ep, self.cb\_cube.eo)

elif self.rot == 1: # صرف با چرخش 120 درجه

cb = cubie.CubieCube(sy.symCube[32].cp, sy.symCube[32].co, sy.symCube[32].ep, sy.symCube[32].eo)

cb.multiply(self.cb\_cube)

cb.multiply(sy.symCube[16])

elif self.rot == 2: # صرف با چرخش 240 درجه

cb = cubie.CubieCube(sy.symCube[16].cp, sy.symCube[16].co, sy.symCube[16].ep, sy.symCube[16].eo)

cb.multiply(self.cb\_cube)

cb.multiply(sy.symCube[32])

if self.inv == 1: # مکعب معکوس

tmp = cubie.CubieCube()

cb.inv\_cubie\_cube(tmp)

cb = tmp

self.co\_cube = coord.CoordCube(cb) # مکعب چرخانده شده / معکوس در نمایش مختصات

dist = self.co\_cube.get\_depth\_phase1()

for togo1 in range(dist, 20): # تعمیق تکراری ، راه حلحداقل حرکت های دور دارد

self.sofar\_phase1 = []

self.search(self.co\_cube.flip, self.co\_cube.twist, self.co\_cube.slice\_sorted, dist, togo1)

#################################End class SolverThread#################################################################

def solve(cubestring, max\_length=20, timeout=3):

""" مکعبی را که با رشته تعریف مکعب تعریف شده حل کنید

: param cubestring: قالب رشته در کلاس Facelet تعریف شده در پرونده enums.py آورده شده است

: param max\_length: اگر مانوری با طول <= max\_length پیدا شود ، تابع برمی گردد

: param timeout: در صورت اتمام تابع ، بهترین راه حل موجود تاکنون بازگردانده شده است. اگر پیدا نشده است

هر راه حلی هنوز محاسبه تا زمان ظهور اولین راه حل ادامه دارد.

"""

#چاپ("Cube string : ", cubestring)

fc = face.FaceCube()

s = fc.from\_string(cubestring)

if s != cubie.CUBE\_OK:

return s # خطا در مکعب facelet

cc = fc.to\_cubie\_cube()

s = cc.verify()

if s != cubie.CUBE\_OK:

return s # خطا در مکعب

my\_threads = []

s\_time = time.monotonic()

# این متغیرهای قابل تغییر توسط هر شش رشته اصلاح می شوند

s\_length = [999]

solutions = []

terminated = thr.Event()

terminated.clear()

syms = cc.symmetries()

if len(list({16, 20, 24, 28} & set(syms))) > 0: # ما دارای یک تقارن چرخشی در امتداد یک مورب طولانی هستیم

tr = [0, 3] # so we search only one direction and the inverse

else:

tr = range(6) # این به معنی جستجو در 3 جهت + مکعب معکوس است

if len(list(set(range(48, 96)) & set(syms))) > 0: # مقداری ضد تقارن داریم بنابراین معکوس ها را جستجو نمی کنیم

tr = list(filter(lambda x: x < 3, tr))

for i in tr:

th = SolverThread(cc, i % 3, i // 3, max\_length, timeout, s\_time, solutions, terminated, [999])

my\_threads.append(th)

th.start()

for t in my\_threads:

t.join() # صبر کنید تا تمام نخ ها به پایان برسد

s = ''

if len(solutions) > 0:

for m in solutions[-1]: # آخرین راه حل کوتاهترین راه حل است s += m.name + ‘ ‘

return s

**نتایج:**

در هنگام اجرای برنامه می پرسد که چند تکرار و چند میگو.

Krill Herd Optimization

Enter the number of iterations : 5

Enter the number of krills : 5

Type M for manual scramble or R for random scrabmle : R

B2 L' R F B R2 D2 F U R F U R2 D L2 F2 U R D B' L' U' F2 D2 R' U2 L2 R D' [29]

Scramble : B2 L' R F B R2 D2 F U R F U R2 D L2 F2 U R D B' L' U' F2 D2 R' U2 L2 R D'

Initial cube orientation: brbwygybwryrrobgyogwggggwrobbygwowwbwrrbroowryyoyboyog

Initial Fitnees : 20

Maximum Iteration : 5

Total Population : 5

Iteration : 0

\*\*\*\*\*\*\*\*\*\* Edge Selection Stage \*\*\*\*\*\*\*\*\*\*

Krill : 0 Orientation : brbwygybwryrrobgyogwggggwrobbygwowwbwrrbroowryyoyboyog Current Fitnees : 20 Method : Greedy Move : D' New Fitnees : 19

Krill : 1 Orientation : brbwygybwryrrobgyogwggggwrobbygwowwbwrrbroowryyoyboyog Current Fitnees : 20 Method : Greedy Move : U New Fitnees : 19

Krill : 2 Orientation : brbwygybwryrrobgyogwggggwrobbygwowwbwrrbroowryyoyboyog Current Fitnees : 20 Method : Greedy Move : U New Fitnees : 19

Krill : 3 Orientation : brbwygybwryrrobgyogwggggwrobbygwowwbwrrbroowryyoyboyog Current Fitnees : 20 Method : Greedy Move : F New Fitnees : 19

Krill : 4 Orientation : brbwygybwryrrobgyogwggggwrobbygwowwbwrrbroowryyoyboyog Current Fitnees : 20 Method : Greedy Move : D' New Fitnees : 19

brbwygybwryrrobyoggwgggggyoyobbwwbgwwrrbrowroyyoyboowr : [0, 4]

ywbbyrwgbyyorobgyoryrgggwrobbygwowwbgwgbroowrwrryboyog : [1, 2]

brbwygroryyrbobwyowggrgwogggrrgwowwbwrbbrbowyyyoyboyog : [3]

\*\*\*\*\*\*\*\*\*\* Killing Stage \*\*\*\*\*\*\*\*\*\*

Global Minimum : 19

Fitnees Threshold : 28

No krill was killed

brbwygybwryrrobyoggwgggggyoyobbwwbgwwrrbrowroyyoyboowr : [0, 4]

ywbbyrwgbyyorobgyoryrgggwrobbygwowwbgwgbroowrwrryboyog : [1, 2]

brbwygroryyrbobwyowggrgwogggrrgwowwbwrbbrbowyyyoyboyog : [3]

\*\*\*\*\*\*\*\*\*\* Reproduction Stage \*\*\*\*\*\*\*\*\*\*

No reproduction as no krill was killed.

\*\*\*\*\*\*\*\*\*\* Mutation Stage \*\*\*\*\*\*\*\*\*\*

Threshold : 28

Krill : 0 Fitnees : 19 No mutation

Krill : 1 Fitnees : 19 No mutation

Krill : 2 Fitnees : 19 No mutation

Krill : 3 Fitnees : 19 No mutation

Krill : 4 Fitnees : 19 No mutation

Iteration : 1

\*\*\*\*\*\*\*\*\*\* Edge Selection Stage \*\*\*\*\*\*\*\*\*\*

Krill : 0 Orientation : brbwygybwryrrobyoggwgggggyoyobbwwbgwwrrbrowroyyoyboowr Current Fitnees : 19 Method : Greedy Move : L2 New Fitnees : 18

Krill : 1 Orientation : ywbbyrwgbyyorobgyoryrgggwrobbygwowwbgwgbroowrwrryboyog Current Fitnees : 19 Method : Probability Move : D New Fitnees : 20

Krill : 2 Orientation : ywbbyrwgbyyorobgyoryrgggwrobbygwowwbgwgbroowrwrryboyog Current Fitnees : 19 Method : Probability Move : D' New Fitnees : 20

Krill : 3 Orientation : brbwygroryyrbobwyowggrgwogggrrgwowwbwrbbrbowyyyoyboyog Current Fitnees : 19 Method : Greedy Move : L New Fitnees : 18

Krill : 4 Orientation : brbwygybwryrrobyoggwgggggyoyobbwwbgwwrrbrowroyyoyboowr Current Fitnees : 19 Method : Greedy Move : R2 New Fitnees : 18

yrbbygbbwryrrobyogrwgoggoyobobwwwygworworbrrwyygybgowg : [0]

ywbbyrwgbyyorobwroryrgggowrwgbwwbboygwgbroyogwrrybogyo : [1]

ywbbyrwgbyyorobyogryrggggyoyobbwwbgwgwgbrowrowrryboowr : [2]

grboygooryyrbobwyobggwgwrggwrrrwoowbobwwrrybbyywybgyog : [3]

brbwywybwgoyborryrgwoggygyyyobbwgbgwwrrbrowrooyogbogwr : [4]

\*\*\*\*\*\*\*\*\*\* Killing Stage \*\*\*\*\*\*\*\*\*\*

Global Minimum : 18

Fitnees Threshold : 27

No krill was killed

yrbbygbbwryrrobyogrwgoggoyobobwwwygworworbrrwyygybgowg : [0]

ywbbyrwgbyyorobwroryrgggowrwgbwwbboygwgbroyogwrrybogyo : [1]

ywbbyrwgbyyorobyogryrggggyoyobbwwbgwgwgbrowrowrryboowr : [2]

grboygooryyrbobwyobggwgwrggwrrrwoowbobwwrrybbyywybgyog : [3]

brbwywybwgoyborryrgwoggygyyyobbwgbgwwrrbrowrooyogbogwr : [4]

\*\*\*\*\*\*\*\*\*\* Reproduction Stage \*\*\*\*\*\*\*\*\*\*

No reproduction as no krill was killed.

\*\*\*\*\*\*\*\*\*\* Mutation Stage \*\*\*\*\*\*\*\*\*\*

Threshold : 27

Krill : 0 Fitnees : 18 No mutation

Krill : 1 Fitnees : 20 No mutation

Krill : 2 Fitnees : 20 No mutation

Krill : 3 Fitnees : 18 No mutation

Krill : 4 Fitnees : 18 No mutation

Iteration : 2

\*\*\*\*\*\*\*\*\*\* Edge Selection Stage \*\*\*\*\*\*\*\*\*\*

Krill : 0 Orientation : yrbbygbbwryrrobyogrwgoggoyobobwwwygworworbrrwyygybgowg Current Fitnees : 18 Method : Probability Move : L2 New Fitnees : 20

Krill : 1 Orientation : ywbbyrwgbyyorobwroryrgggowrwgbwwbboygwgbroyogwrrybogyo Current Fitnees : 20 Method : Greedy Move : D' New Fitnees : 19

Krill : 2 Orientation : ywbbyrwgbyyorobyogryrggggyoyobbwwbgwgwgbrowrowrryboowr Current Fitnees : 20 Method : Greedy Move : R2 New Fitnees : 19

Krill : 3 Orientation : grboygooryyrbobwyobggwgwrggwrrrwoowbobwwrrybbyywybgyog Current Fitnees : 18 Method : Probability Move : B2 New Fitnees : 20

Krill : 4 Orientation : brbwywybwgoyborryrgwoggygyyyobbwgbgwwrrbrowrooyogbogwr Current Fitnees : 18 Method : Probability Move : L' New Fitnees : 20

brbwygybwryrrobyoggwgggggyoyobbwwbgwwrrbrowroyyoyboowr : [0]

ywbbyrwgbyyorobgyoryrgggwrobbygwowwbgwgbroowrwrryboyog : [1]

ywbbywwgwgoyboroyyryoggygywyobbwrbgbgwgbrowroorrgborwr : [2]

bwooygooryyybowwyobggwgwrggwrrrwobrgobwbrrrbbgoygbywyy : [3]

grbgywgbwgoyborryrywobgybyyrobowgogwroorrrwbwoyygbwgwb : [4]

\*\*\*\*\*\*\*\*\*\* Killing Stage \*\*\*\*\*\*\*\*\*\*

Global Minimum : 19

Fitnees Threshold : 28

No krill was killed

brbwygybwryrrobyoggwgggggyoyobbwwbgwwrrbrowroyyoyboowr : [0]

ywbbyrwgbyyorobgyoryrgggwrobbygwowwbgwgbroowrwrryboyog : [1]

ywbbywwgwgoyboroyyryoggygywyobbwrbgbgwgbrowroorrgborwr : [2]

bwooygooryyybowwyobggwgwrggwrrrwobrgobwbrrrbbgoygbywyy : [3]

grbgywgbwgoyborryrywobgybyyrobowgogwroorrrwbwoyygbwgwb : [4]

\*\*\*\*\*\*\*\*\*\* Reproduction Stage \*\*\*\*\*\*\*\*\*\*

No reproduction as no krill was killed.

\*\*\*\*\*\*\*\*\*\* Mutation Stage \*\*\*\*\*\*\*\*\*\*

Threshold : 28

Krill : 0 Fitnees : 20 No mutation

Krill : 1 Fitnees : 19 No mutation

Krill : 2 Fitnees : 19 No mutation

Krill : 3 Fitnees : 20 No mutation

Krill : 4 Fitnees : 20 No mutation

Iteration : 3

\*\*\*\*\*\*\*\*\*\* Edge Selection Stage \*\*\*\*\*\*\*\*\*\*

Krill : 0 Orientation : brbwygybwryrrobyoggwgggggyoyobbwwbgwwrrbrowroyyoyboowr Current Fitnees : 20 Method : Greedy Move : L2 New Fitnees : 18

Krill : 1 Orientation : ywbbyrwgbyyorobgyoryrgggwrobbygwowwbgwgbroowrwrryboyog Current Fitnees : 19 Method : Probability Move : U New Fitnees : 20

Krill : 2 Orientation : ywbbywwgwgoyboroyyryoggygywyobbwrbgbgwgbrowroorrgborwr Current Fitnees : 19 Method : Probability Move : L2 New Fitnees : 19

Krill : 3 Orientation : bwooygooryyybowwyobggwgwrggwrrrwobrgobwbrrrbbgoygbywyy Current Fitnees : 20 Method : Greedy Move : B2 New Fitnees : 18

Krill : 4 Orientation : grbgywgbwgoyborryrywobgybyyrobowgogwroorrrwbwoyygbwgwb Current Fitnees : 20 Method : Greedy Move : R' New Fitnees : 19

yrbbygbbwryrrobyogrwgoggoyobobwwwygworworbrrwyygybgowg : [0]

wbygywbrbwrrrobgyoyyogggwrobbygwowwbryrbroowrgwgyboyog : [1]

ywbbywbgwgoyboroyyryoogyrywyobbwrwgborworbgwgorggbgrwr : [2]

grboygooryyrbobwyobggwgwrggwrrrwoowbobwwrrybbyywybgyog : [3]

grggyggboyrrooygbrywbbgwbywrooowyogyroorrrwbwwyygbwbwb : [4]

\*\*\*\*\*\*\*\*\*\* Killing Stage \*\*\*\*\*\*\*\*\*\*

Global Minimum : 18

Fitnees Threshold : 27

No krill was killed

yrbbygbbwryrrobyogrwgoggoyobobwwwygworworbrrwyygybgowg : [0]

wbygywbrbwrrrobgyoyyogggwrobbygwowwbryrbroowrgwgyboyog : [1]

ywbbywbgwgoyboroyyryoogyrywyobbwrwgborworbgwgorggbgrwr : [2]

grboygooryyrbobwyobggwgwrggwrrrwoowbobwwrrybbyywybgyog : [3]

grggyggboyrrooygbrywbbgwbywrooowyogyroorrrwbwwyygbwbwb : [4]

\*\*\*\*\*\*\*\*\*\* Reproduction Stage \*\*\*\*\*\*\*\*\*\*

No reproduction as no krill was killed.

\*\*\*\*\*\*\*\*\*\* Mutation Stage \*\*\*\*\*\*\*\*\*\*

Threshold : 27

Krill : 0 Fitnees : 18 No mutation

Krill : 1 Fitnees : 20 No mutation

Krill : 2 Fitnees : 19 No mutation

Krill : 3 Fitnees : 18 No mutation

Krill : 4 Fitnees : 19 No mutation

Iteration : 4

\*\*\*\*\*\*\*\*\*\* Edge Selection Stage \*\*\*\*\*\*\*\*\*\*

Krill : 0 Orientation : yrbbygbbwryrrobyogrwgoggoyobobwwwygworworbrrwyygybgowg Current Fitnees : 18 Method : Probability Move : B2 New Fitnees : 19

Krill : 1 Orientation : wbygywbrbwrrrobgyoyyogggwrobbygwowwbryrbroowrgwgyboyog Current Fitnees : 20 Method : Greedy Move : U' New Fitnees : 19

Krill : 2 Orientation : ywbbywbgwgoyboroyyryoogyrywyobbwrwgborworbgwgorggbgrwr Current Fitnees : 19 Method : Greedy Move : R' New Fitnees : 18

Krill : 3 Orientation : grboygooryyrbobwyobggwgwrggwrrrwoowbobwwrrybbyywybgyog Current Fitnees : 18 Method : Probability Move : F New Fitnees : 18

Krill : 4 Orientation : grggyggboyrrooygbrywbbgwbywrooowyogyroorrrwbwwyygbwbwb Current Fitnees : 19 Method : Greedy Move : F2 New Fitnees : 18

wgybygbbwryrrooyoorwgoggoyobobwwwbrygrwbrbrrwgwogbygyy : [0]

ywbbyrwgbyyorobgyoryrgggwrobbygwowwbgwgbroowrwrryboyog : [1]

ywrbygbgoyryooygborybogwrywyoobwywgworworbgwgbrgrbgbwr : [2]

grboygbrwoyroobryorwbggggwgwbyrwoowbobwwrrybryywybgyog : [3]

grggygoorwrrroyobrwybwgbbwyobgowyogyrogrrowbywyygbwbwb : [4]

\*\*\*\*\*\*\*\*\*\* Killing Stage \*\*\*\*\*\*\*\*\*\*

Global Minimum : 18

Fitnees Threshold : 27

No krill was killed

wgybygbbwryrrooyoorwgoggoyobobwwwbrygrwbrbrrwgwogbygyy : [0]

ywbbyrwgbyyorobgyoryrgggwrobbygwowwbgwgbroowrwrryboyog : [1]

ywrbygbgoyryooygborybogwrywyoobwywgworworbgwgbrgrbgbwr : [2]

grboygbrwoyroobryorwbggggwgwbyrwoowbobwwrrybryywybgyog : [3]

grggygoorwrrroyobrwybwgbbwyobgowyogyrogrrowbywyygbwbwb : [4]

\*\*\*\*\*\*\*\*\*\* Reproduction Stage \*\*\*\*\*\*\*\*\*\*

No reproduction as no krill was killed.

\*\*\*\*\*\*\*\*\*\* Mutation Stage \*\*\*\*\*\*\*\*\*\*

Threshold : 27

Krill : 0 Fitnees : 19 No mutation

Krill : 1 Fitnees : 19 No mutation

Krill : 2 Fitnees : 18 No mutation

Krill : 3 Fitnees : 18 No mutation

Krill : 4 Fitnees : 18 No mutation

Cube orientation: Yellow on top and green on front.

Reading order goes: Up, Right, Front, Down, Left, Back.

Scramble: B2 L' R F B R2 D2 F U R F U R2 D L2 F2 U R D B' L' U' F2 D2 R' U2 L2 R D'

Initial cube orientation : brbwygybwryrrobgyogwggggwrobbygwowwbwrrbroowryyoyboyog

Solution was not found by any krill

The total number of krill that found solution: 0 / 5

Original Solution Length: 20

Average solution length: NIL

Iteration at which the first solution was found: NIL

Iteration at which the last krill found the solution: NIL

Average iterations it takes for the next krill to reach solved state: NIL

Total time for loop execution : 2693.979466199875