به نام خدا

تمرین اول

ستایش کولوبندی
97522202

در این تمرین با توجه به الگوریتم های سرچ پازل water sort را حل میکنیم.

بخش اول :

با استفاده از dfs سوال را حل میکنیم.

برای حل سوال نیاز داریم ببینم بین کدام دو لوله میتوان آب را جابه جا کرد

```
    def emkan_jabejayi(self,current_state,moves):
        # jaye khali dashte bashe   # va age balataring rang maghsad = mabda
        if (len(current_state[moves[1]])<self.ws_game.NColorInTube and len(current_state[moves[0]])>0 ):
            if(len(current_state[moves[1]])==0):  # bekhter indexout of range joda ke age khali bud maghsad ke dige okaye
                return True
            elif(current_state[moves[0]][-1]==current_state[moves[1]][-1]) :
                return True
            else:
                return False
        else:
            return False

```

این الگوریتم با چک کردن دو شرطی که در کامنت گفته شده و جزو قوانین بازی است به عنوان خروجی بسته به اینکه بتوانیم آب را از مبدا گفته شده move 0 به مقصد jmove 1 بریزیم true یا false بر میگرداند.

تابع solve به صورت بازگشتی کار میکند ابتدا چک میکنیم که اگر جواب پیدا شده return کن

```
def solve(self, current_state):

    c_state=copy.deepcopy(current_state)

    if self.ws_game.check_victory(c_state) or self.solution_found:
        self.solution_found=True
        return
    else:
```

از current state به خاطر این کپی گرفتم که در صورت تغییر current state به صورت گرافیکی هم رنگ لوله ها سریع عوض میشد.

سپیس با دو for تو در تو روی تمام شماره لوله ها برای جا به جایی پیمایش میکنیم

```
40                for i in range(len(c_state)) :
41                    if self.solution_found:
42                        return
43                    for j in range(i+1,len(c_state)) :
44                        if self.solution_found:
45                            return
46
```

```
48                    if(self.emkan_jabejayi(c_state,(i,j))):
49
50                        temp=copy.deepcopy(c_state)
51                        while(self.emkan_jabejayi(temp,(i,j))):
52
53                            temp[j].append(temp[i][-1]) # akhe move curr
54                            temp[i].pop()
55                        if not (str(temp) in self.visited_tubes):
56                            c_state=temp
57                            self.moves.append((i,j))
58                            self.visited_tubes.add(str(c_state))
59                            if (self.ws_game.check_victory(c_state)):
60                                self.solution_found=True
61                                return
62                            else:
63                                self.solve(c_state)
```

در صورتی که امکان جابه جایی بین دو لوله باشد (بین راس های درختمان یالی باشد):
خط 51 برای اینه که اگه دو تا رنگ عین هم بالا بود و لوله مقصد هم دو تا جا داشت هر دو رو با هم جابه جا کنیم
بعد state ای که در آن آب ها جابهجا شدند رو به عنوان temp نگه میداریم که چک کنیم اگه قبلا به این state نیومدیم بریم توش و بعد به لیست state های دیده شده و لیست حرکات اطلاعات را اضافه میکنیم و در صورتی که state جدید همان goal باشد return میکنیم و self.solution_found هم true میکنیم که از بقیه ی تابع های بازگشتی که داخل هم صدا زده شدند هم return کنیم در عیر این صورت تابع solve را بازگشتی صدا میزنیم.

```
65                  if(self.emkan_jabejayi(c_state,(j,i))):
66                      temp2=copy.deepcopy(c_state)
67
68                      while(self.emkan_jabejayi(temp2,(j,i))):
69                          temp2[i].append(temp2[j][-1]) # akhe move curre
70                          temp2[j].pop()
71
72                      if not (str(temp2) in self.visited_tubes):
73                          c_state=temp2
74                          self.moves.append((j,i))
75                          self.visited_tubes.add(str(c_state))
76                          if (self.ws_game.check_victory(c_state)):
77                              self.solution_found=True
78
79                              return
80                          else:
81                              self.solve(c_state)
82
```

این if هم همان کار if قبلی رو میکنه فقط برای این گذاشتم که وقتی 1,2چک میشه 1 و 2 هم چک بشه

```
82
83              if self.solution_found:
84                  return
85              elif len(self.moves)>0 : # hanuz momkene javab das
86                  #print(" BACK TRACKING ")
87                  (a , b)= self.moves.pop()
88                  c_state[a].append(c_state[b][-1])
89                  c_state[b].pop() if c_state[b] else None
90                  self.solve(c_state)
91              else:
92                  self.solution_found=False
93                  return
94          pass
```

این قسمت برای backtracing که اگه به راسی رسیدیم که روی همه ی احتمالات جابه جایی پیمایش کردیم ولی به جواب نرسیدیم به یک راس قبل تر برگردیم که چون آخرین حرکت ما آخرین عضو move با move.pop جرکت را استراج کرده و به state قبلی برمیگردیم.

بخش دوم :

در این بخش میخواهیم با استفاده از a* سوال را حل کنیم.

برای اینکار به یک heuristic مناسب نیاز داریم

```
96      def heuristic(self,current_state):
97          count=0
98          for i in range(len(current_state)):
99              if len(current_state[i])>1 :
100                 tcount=current_state[i][0]  #tube count
101                 for j in range(1,len(current_state[i])):
102                     if not(current_state[i][j]==tcount):
103                         count+=1  # vali akharesh bad didan har
104                         tcount=(current_state[i][j])
105
```

هیوریستیک پیاده سازی شده برای هر استیت برای هر لوله تعداد رنگ هایی را که بالاتر از رنگ اول هستند و رنگشون با رنگ قبلیشون یکی نیست را با هم جمع میکند و به عنوان خروجی برمیگرداند چون حداقل حرکاتی که مطمئنیم نیازه برداشتن رنگ های بالایی است.

```
def optimal_solve(self, current_state):
    c_state=copy.deepcopy(current_state)
    hs = {str(c_state):(self.heuristic(c_state))} # dectionary h ha
    gs = {str(c_state):0} # dectionary g ha
    pq=[]
    heapq.heappush(pq,((hs[str(c_state)]+gs[str(c_state)]), (c_state,[])))
    s_move=[]
    state=[]
    b=0
```

در این جا مانند تابع solve ابتدا از current_state کپی گرفتم برای این حل ما به دو دیکشنری یکی برای هیوریستیک های هر استیت و یکی هم برای g های هر استیت نیاز داریم و به یک priority queue که کل اطلاعات راس های frontier را در خودش ذخیره کنید.

```
120         while len(pq)>0:
121 
122             temp=heapq.heappop(pq)
123             f=temp[0]
124             state=temp[1][0]
125             s_move=temp[1][1]
126             last_g =gs[str(state)]
127             self.visited_tubes.add(str(state))
128             if(self.ws_game.check_victory(state)):
129                 self.solution_found=True
130                 self.moves=s_move
131                 return
132             elif not self.solution_found:
133                 for i in range(len(state)) :
134                     for j in range(i+1,len(state)) :
135                         state_t=copy.deepcopy(state)
```

در این قسمت مطابق الگوریتم bfs جلو میرویم
تا وقتی که نود های frontier تمام نشدند یا تا وقتی به نتیجه نرسیدیم ادامه میدهیم. در هر مرحله بهترین frontier موجود را از صف خارج میکنیم و اطلاعات راس خارج شده را در متغیر های s_move last_g و دیگر متغیر ها ذخیره میکنیم.
و سپس روی تمام همسایه هایش و تمام دو لوله بین تعداد لوله ها پیمایش میکنیم.

```
if(self.emkan_jabejayi(state_t,(i,j))):
    temp=copy.deepcopy(state_t)

    while(self.emkan_jabejayi(temp,(i,j))):

        temp[j].append(temp[i][-1]) # akhe move current state_t ro avaz nemikone
        temp[i].pop()
    new_move= s_move+ [(i,j)]
    new_g=(int(last_g)+1)
    bude = str(temp) in self.visited_tubes

    if not bude or (bude and new_g<gs[str(temp)]) :   # age bude bashe h ha yeki
        state_t=temp
        new_f=self.heuristic(temp)
        gs[str(state_t)]=new_g
        hs[str(state_t)]=new_f
        heapq.heappush(pq,((new_f+ new_g), (state_t,new_move)))
        self.visited_tubes.add(str(state_t))
```

شبیه به تابه solve این جا هم لوله هایی که امکان جابه جایی داشته باشند را پیدا میکنیم و در یک متغیر temp میریزم تا اول چک کنیم قبلا این متغیر را دیده ایم یا نه از انجایی که من داخل self.visited_tubes فقط استیت را ذخیره میکنم نه move ها رو پس ممکنه بخواهیم به state تکراری برویم در صورتی امکان این موضوع هست که از راه بهتری به اون state رسیده باشیم یعنی g state کم تر باشد.

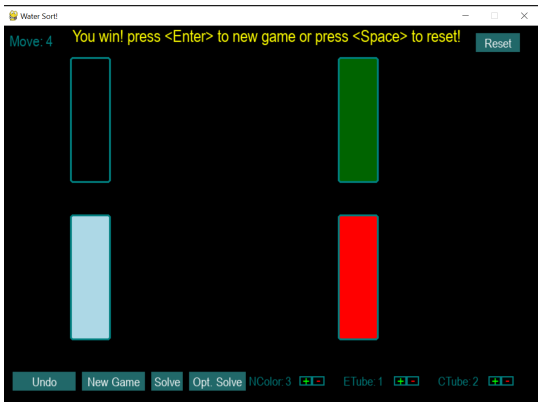
```
if(self.emkan_jabejayi(state_t,(j,i))):
    temp=copy.deepcopy(state_t)
    while(self.emkan_jabejayi(temp,(j,i))):

        temp[i].append(temp[j][-1]) # akhe move current state_t ro avaz nemikone
        temp[j].pop()
    new_move= s_move+ [(j,i)]
    new_g=(int(last_g)+1)
    bude = str(temp) in self.visited_tubes
    if not bude or (bude and new_g<gs[str(temp)]) :
        state_t=temp
        new_f=self.heuristic(temp)
        gs[str(state_t)]=new_g
        hs[str(state_t)]=new_f
        heapq.heappush(pq,((new_f+ new_g), (state_t,new_move)))
        self.visited_tubes.add(str(state_t))
```

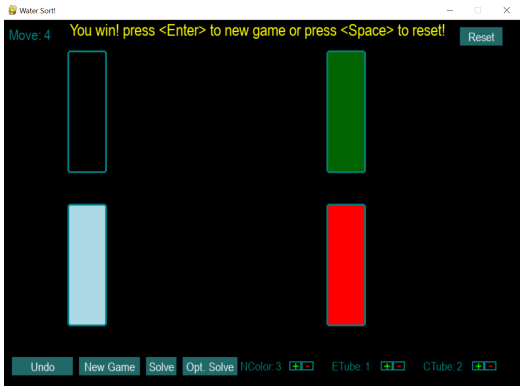
این if هم شبیه به if دوم تابع solve است و به همان دلیل نوشته شده.

در نهایت تعداد move ها را برای هر کدام از دو الگوریتم برای مثال های مختلف در پایین مشاهده میکنیم:

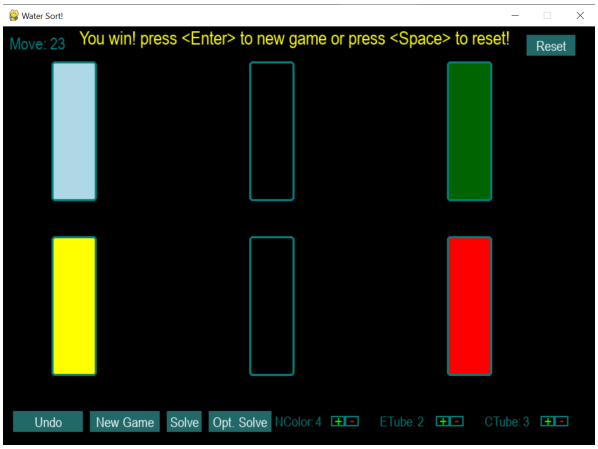
Solve:



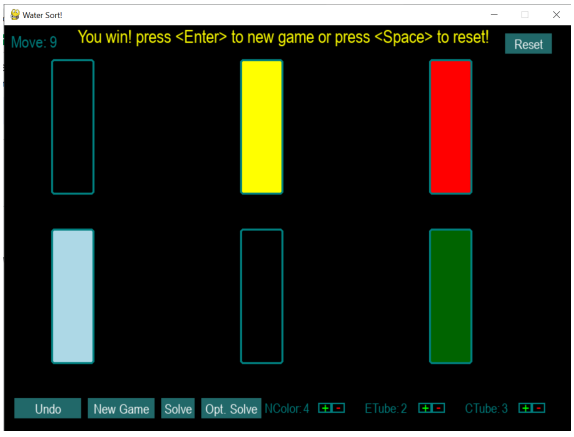
Optimal:

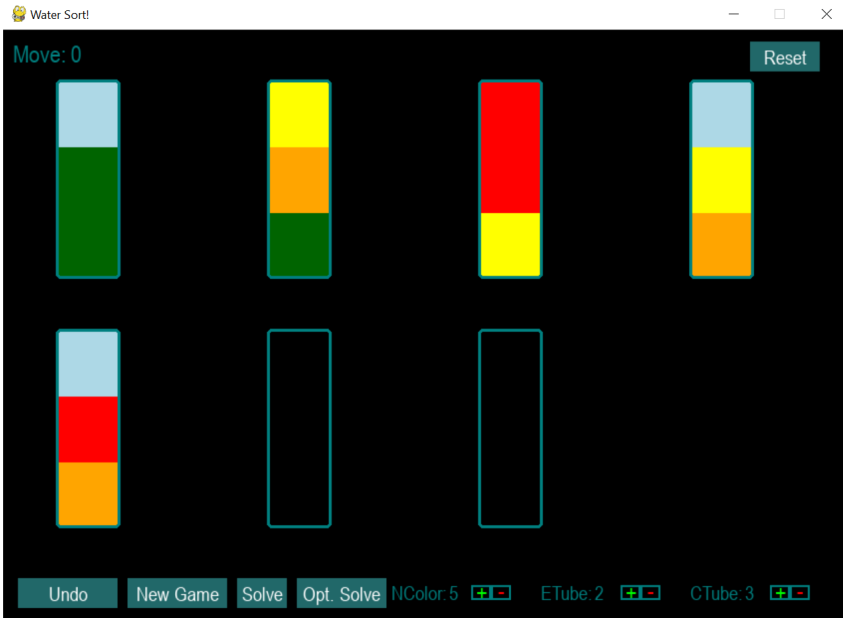


Solve:

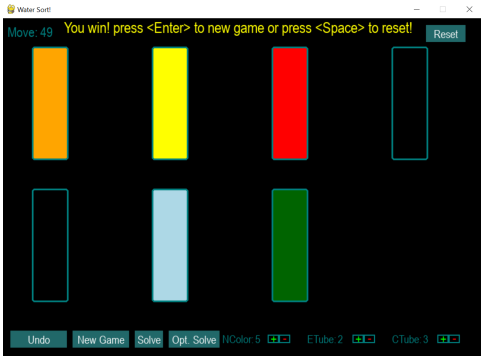


Optimal solve:





Solve:



Optimal: