کیانا خیلی ۴۰۲۳۳۰۹۶

هدف این تمرین حل سازی و پیاده سازی مسئله برج هانوی است در قالب فضای حالت (state space) است. می‌دانیم سه تا کلاس طراحی می‌کنیم ۱- کلاس state: نمایش دهنده یک حالت از مسئله در فضای جستجو است و می‌تواند برای هر مسئله که نیاز به جستجو دارد گسترش یابد ۲- کلاس problem: این کلاس مشخصات کلی یک مسئله را تعریف می‌کند مانند حالت اولیه، تست هدف، تابع جانشین و تابع هزینه و تعریف حالت هدف ۳- کلاس TowerofHanoi: پیاده سازی اختصاصی مسئله برج هانوی با ارث بری از Problem با استفاده از متدهای این کلاس حالت تولید شده و هیوریستیک‌های ممکن برای جابجایی دیسک‌ها از میله اول به میله سوم نمایش داده می‌شود

مسئله برج هانوی شامل سه میله (A و B و C) و تعدادی دیسک با اندازه‌های مختلف است در ابتدا تمام دیسک‌ها روی میله A به ترتیب از بزرگ به کوچک قرار دارند هدف این است که تمام دیسک‌ها به میله C منتقل شوند به طوری که ۱- در هر حرکت فقط یک دیسک جابجا شود. ۲- هیچ دیسک بزرگ‌تری روی دیسک کوچک‌تر قرار نمی‌گیرد هر وضعیت از چیدمان دیسک‌ها یک state در فضای جستجو محسوب می‌شود با اعمال حرکات مجاز از حالت اولیه به حالت هدف نزدیک می‌شویم

ساختار کلاس‌ها:

۱- کلاس state کلاس انتزاعی برای نمایش یک حالت از مسئله ویژگی‌های متداول مقایسه دو حالت __eq__
نمایش به صورت رشته __str__

۲- کلاس problem: کلاس انتزاعی که ساختارهای یک مسئله جست‌وجو را مشخص می‌کند شامل
initial_state حالت اولیه
goal_test بررسی رسیدن به هدف
successors تولید حالت‌های جانشین

۳- کلاس TowerofHanoi کلاس اصلی که از Problem ارث‌بری می‌کند و منطق برج هانوی را

پیاده‌سازی می‌نماید حقه‌ها: بررسی انتقال تمام دیسک‌ها به میله مقصد goal_test

تولید تمام حرکت‌های ممکن: successors

تولید فضای حالت تا تعداد مشخصی از وضعیت‌ها generator_state_space

```
class State(ABC):
    def __eq__(self, other):
        pass

    def __str__(self):
        pass

class Problem(ABC):
    def __init__(self, initial_state):
        self.initial_state = initial_state

    def goal_test(self, state):
        pass

    def successors(self, state):
        pass

class HanoiState(State):
    def __init__(self, pegs):
        self.pegs = pegs

    def __eq__(self, other):
        return str(self.pegs)
```

```
def copy(self):
    return HanoiState(deepcopy(self.pegs))

class TowerofHanoi(problem):

    def __init__(self, n):
        initial_pegs = [list(range(n, 0, -1)), [], []]
        super().__init__(HanoiState(initial_pegs))
        self.n = n

    def goal_test(self, state):
        return len(state.pegs[2]) == self.n

    def successors(self, state):

        successors = []
        for i in range(3):
            if not state.pegs[i]:
                continue
            disk = state.pegs[i][-1]
            for j in range(3)
                if i != j and (not state.pegs[j] or
                                state.pegs[j][-1]
                    new_state = state.copy()
                    new_state.pegs[i].pop()
                    new_state.pegs[j].append(disk)
                    successors.append(new_state)
        return successors

    def generate_state_space(self, limit=100):

        visited = set()
        f = [self.initial_state]

        all_state = []
```

```
while f and len(all_states) < limit:
    current = f.pop()
    state_str = str(current.pegs)
    if state_str in visited:
        continue
    visited.add(state_str)
    all_states.append(current)
    for s in self.successors(current):
        f.append(s)
return all_states
```