تعریف مسئله

مسئله N_QUEEN یعنی N وزیر را در یک صفحه شطرنج N x N طوری بچیند که حمله به یکدیگر برای هر وزیر غیرممکن باشد.

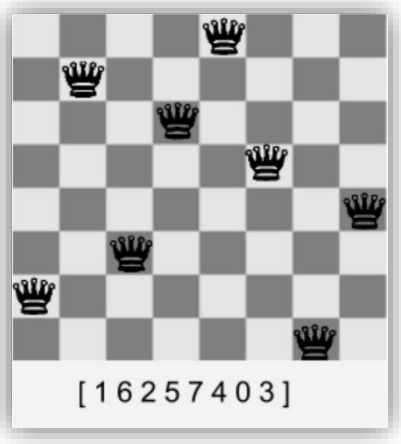
برای اینکه این مسئله را با الگوریتم ژنتیک حل کنیم مراحل زیر لازم است:

1) *یک کروموزوم تصادفی تولید کنیم*
2)محاسبهfitness
3)اگرfitness با fmaxبرابر نیست
4)نقطه تلاقی وسط کروموزوم (crossover)
5)جهش(mutation)
6)کروموزوم جدید به جمعیت اضافه شده است
مرحله 2تا6 را تکرار کن تا یک کروموزوم با مقدارfitness=maxfitness پیدا شود

برای نشان دادن یک کروموزوم به این صورت عمل میکنیم:

یک آرایه تک بعدی ایجاد می‌کنیم که به تعداد ستون‌های صفحه شطرنج عنصر دارد.

هر عنصر از این آرایه نشان می‌دهد که وزیر در کدام سطر از آن ستون قرار دارد.



به عنوان مثال اگر مسئله ۸ وزیر را در نظر بگیریم،

آرایه تک بعدی باید دارای ۸ عنصر باشد.

آرایه ی شکل مقابل درنظر بگیرید

[1,6,2,5,7,4,0,3]

مقدار 1 در اولین عنصر آرایه گویای این مطلب است
که در ستون اول صفحه شطرنج وزیری در سطر 1ام قرار داده‌ایم.

جمعیت هر نسل می تواند تعداد کروموزوم ها را تعیین کند.

جمعیت اولیه از انتخاب رندومی از کروموزوم ها ایجاد می شود.

الگوریتم‌های ژنتیک ابتدا جمعیت اولیه‌ای تولید کرده و سپس سعی در بهبود این جمعیت دارند

برای مسئله یn وزیر تولید جمعیت به صورت تصادفی خواهد بود.

بدین صورت که وزیرها به‌طور تصادفی روی صفحه شطرنج قرار می‌دهیم.

```
random_chromosome = lambda size: [random.randint(1,size) for _ in range(size)]
random_chromosome.__doc__='making random chromosomes'
```

:fitness

تابع fitness برای محاسبه تعداد جفت وزیرهایی که حمله نمی کنند.

برای یافتن مقدار تابع fitness ازین معادله استفاده میکنیم:

Fitness function = F1+F2+F3+F4+F5+F6+F7+F8

که F1تعداد جفت ملکه های غیرحمله با ملکهQ1و...

ما باید تابع Fitness را برای همه‌ی افراد جمعیت خود(کروموزوم ها) محاسبه کنیم.

```
def fitness(chromosome,maxFitness=None):
    n = len(chromosome)
    if maxFitness==None:
        maxFitness=(n*(n-1))/2
    horizontal_collisions = sum([chromosome.count(queen)-1 for queen in chromosome])/2

    left_diagonal=[0]*(2*n)
    right_diagonal=[0]*(2*n)
    for index,chrom in enumerate(chromosome):
        left_diagonal[index+chrom-1]+=1
        right_diagonal[n-index+chrom-2]+=1

    diagonal_collisions = 0
    for i in range(2*n-1):
        counter = 0
        if left_diagonal[i]  > 1: counter += left_diagonal[i]-1
        if right_diagonal[i] > 1: counter += right_diagonal[i]-1
        diagonal_collisions+= counter / (n-abs(i-n+1))

    return int(maxFitness-(horizontal_collisions + diagonal_collisions))
```

سپس باید احتمال انتخاب از تابع Fitness را محاسبه کنیم.

```
probability=lambda chromosome, fitness,maxFitness=1: fitness(chromosome)/maxFitness
```

در مرحله بعدی ، ما به طور تصادفی دو جفت را برای تولید مثل براساس احتمالاتی که روی مرحله قبل حساب کردیم انتخاب می کنیم.

تابع انتخاب تصادفی:

```
def random_pick(population, probabilities):
    # tmp={tuple(i):j for i,j in zip(population, probabilities)}
    # return list(max(tmp,key=lambda x:tmp[x]))
    r = sum(probabilities)*random.random()
    upto = 0
    for c, w in zip(population, probabilities):
        upto+=w
        if upto>=r: return c
    raise RuntimeError("This is unreachable state :(")
```

:Crossover

یک نقطه تصادفی در طول رشته انتخاب میشود

والدین در این نقطه به دوقسمت میشون

هر فرزند با انتخاب تکه اول از یکی از والدین و تکه دوم از والد دیگر بوجود میاید

```
def reproduce(x, y):
    assert len(x)==len(y)
    '''doing cross_over between two chromosomes'''
    c = random.randint(0,len(x)-1)
    return x[:c]+y[c:]
```

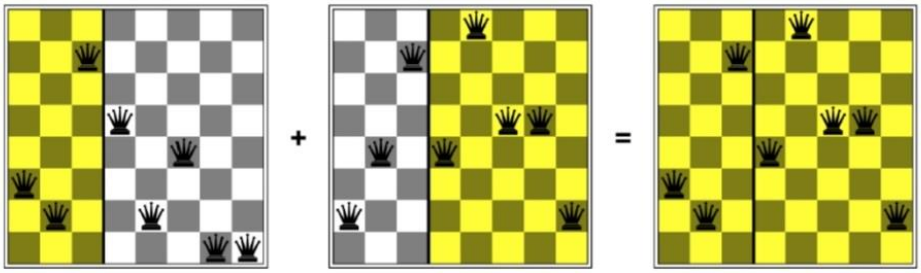
مرحله بعدmutation (جهش):

بصورت تصادفی چند ژن را تغییرمیدهیم

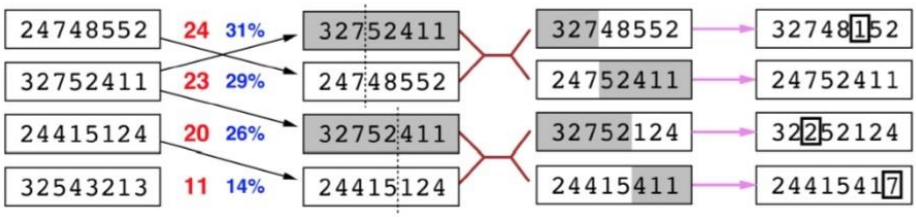
وکروموزوم های جدید بدست امده جمعیت جدید ما خواهد بود

```
def mutate(x):
    '''randomly changing the value of a random index of a chromosome'''
    n = len(x)
    x[random.randint(0,n-1)]=random.randint(1,n)
    return x
```

بنابراین الگوریتم ژنتیک برای حل الگوریتم Queen-8 مانند شکل زیر است.



Generate successors from pairs of states.



در مرحله بعدی ، باید به مرحله 2 (fitness) برگردیم تا تابع fitnessجمعیت به روز شده خود را پیدا کنیم.

مراحل 2-6 تکرار می شوند تا زمانی که کروموزوم موارد زیر را برآورده کند:

مقدار maxfitness == fitness

.

تابع queen_genetic :

ابتدا احتمالات را بااستفاده از تابع probability به دست می اورد.

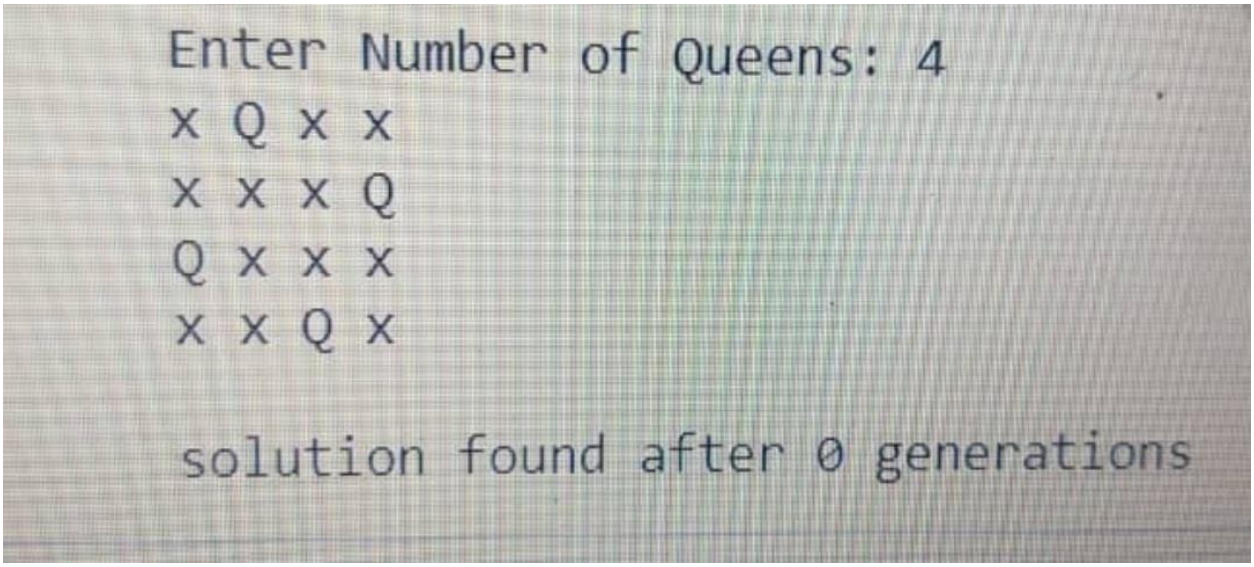
سپس در یک حلقه For بااستفاده از تابع pick_random دو تا از بهترین کروموزم ها را انتخاب کرده .بعد بااستفاده از تابع reproduceدو کروموزوم جدید ایجاد میشود .

در مرحله بعدی یک عدد تصادفی ایجاد کرده با استفاده از random.random()ان را با mutationProbability مقایسه میکنیم .

اگر mutationProbability کمتر از عدد تصدفی ایجاده شده بود , فرزندان ایجاد شده را به تابع mutate جهش میدهیم سپس انها را در لیست population_new قرار میدهیم . این تابع تا زمانی ادامه پیدا میکند که fitness کروموزوم ها به حداکثر خود برسد یا تمام جمعیت را پیمایش کند .در اخر جمعیت جدید را برمیگرداند

```
def genetic_queen(population, fitness,maxFitness,mutationProbability = 0.03):
    new_population = []
    probabilities = [probability(n, fitness,maxFitness) for n in population]
    probabilities = [probability(n, fitness,maxFitness) for n in population]
    for _ in population:
        x = random_pick(population, probabilities) #best chromosome 1
        y = random_pick(population, probabilities) #best chromosome 2
        child = reproduce(x, y) #creating two new chromosomes from the best 2 chromosomes
        if random.random() < mutationProbability:
            child = mutate(child)
        new_population.append(child)
        if fitness(child) == maxFitness: break
    return new_population
```

تست خروجی برای ورودی 4:



فاطمه خدادادی-97143013