1. الف)

برای حل این مسئله با الگوریتم ژنتیک لازم است که هر یک از وسایلی که آقای اهم میتواند بردارد یا برندارد را p(x) نامید. این یعنی اگر مثلا اهم نقاشی را بردارد مقدار p4 یک شده و اگر برندارد 0 میشود. برای انجام اینکار باید ابتدا جمعیت اولیه را تولید کنیم که هر کروموزوم یک رشته ی 5 تایی از 0 و 1 است. جمعیت اولیه میتواند به صورت زیر باشد :

10111 و 11101 : حال باید برای هر دو عضو این جمعیت مقدار برازش را محاسبه کنیم :

دور 1 :
والد اول 10011 : برازش : 5000 + 3000 + 1200 = 9200 وزن : 17
والد دوم 11101 : برازش : 5000 + 1500 + 800 + 1200 = 8500 وزن : 16
بعد از تولید جمعیت اولیه و محاسبه برازش باید از بازترکیب و جهش استفاده کنیم :
فرزند اول : 10101 بعد از جهش بیت 4 ام : 10111 وزن : 21 پس این فرزند حذف میشود.
فرزند دوم : 11011 بعد از جهش بیت 5 ام : 11010 برازش : 5000 + 1500 + 3000 = 9500 وزن :10
حال باید فرزندان تولید شده را جایگزین کنیم فرزند اول که وزنش از 21 بیشتر شده پس جایگزین نمیشود ولی فرزند دوم را با والد دوم که برازش کمتری دارد (استراتژی رقابتی) ، جایگزین میکنیم.
حال باید چک کنیم که شرط توقف دارین یا نه از آنجایی که بهترین برازش تغییر کرده روند را ادامه میدهیم تا به جایی برسیم که بهترین پاسخ تغییری نکند.

دور 2 :
والد اول 10011 : برازش : 5000 + 3000 + 1200 = 9200 وزن : 17
والد دوم 11010 برازش : 5000 + 1500 + 3000 = 9500 وزن :10
برای محاسبه برازش باید از بازترکیب و جهش استفاده کنیم :
فرزند اول : 10010 بعد از جهش بیت 2 و 5 ام : 11011 برازش : 10700 وزن : 18
فرزند دوم : 11011 بعد از جهش بیت 3 و 4 ام : 11101 برازش : 8500 وزن :16
چون یکی از فرزند ها برازش بیشترین برازش را دارد پس با والد اول با برازش کمتر عوض کرده و چون والد دوم برازش بیشتر دارد با فرنزند دوم عوض نمیکنیم. شرط توقف هم نداریم چون به جواب بهتری رسیدیم و ادامه میدهیم.

دور 3 :
والد اول 11011 برازش : 10700 وزن : 18
والد دوم 11010 برازش : 9500 وزن :10
برای محاسبه برازش باید از بازترکیب و جهش استفاده کنیم :
فرزند اول : 11011 بعد از جهش بیت 1 و 3 ام : 01111 برازش : 6500 وزن : 19
فرزند دوم : 11010 بعد از جهش بیت 3 و 5 ام : 11111 وزن :22 پس این فرزند حذف میشود.

فرزند دوم که وزنش از 21 بیشتر شده پس جایگزین نمیشود. فرزند دوم هم از کمترین برازش یعنی والد دوم برازش کمتری دارد پس عوض نمیکنیم. شرط توقف هم نداریم چون هنوز ممکن است به جواب بهتری برسیم و ادامه میدهیم.

دور 4:
والد اول 11011 برازش : 10700 وزن : 18
والد دوم 11010 برازش : 9500 وزن :10
برای محاسبه برازش باید از بازترکیب و جهش استفاده کنیم :
فرزند اول : 11011 بعد از جهش بیت 2 و 3 ام : 10111 وزن : 21 پس این فرزند حذف میشود.
فرزند دوم : 11010 بعد از جهش بیت 4 و 5 ام : 11001 برازش : 7700 وزن :12

فرزند اول که وزنش از 21 بیشتر شده پس جایگزین نمیشود. فرزند دوم هم از کمترین برازش یعنی والد دوم برازش کمتری دارد پس عوض نمیکنیم. شرط توقف هم نداریم چون هنوز ممکن است به جواب بهتری برسیم و ادامه میدهیم.

دور 5 :
والد اول 11011 برازش : 10700 وزن : 18
والد دوم 11010 برازش : 9500 وزن :10
برای محاسبه برازش باید از بازترکیب و جهش استفاده کنیم :
فرزند اول : 11011 بعد از جهش بیت 3 و 5 ام : 11110 برازش : 10400 وزن : 14
فرزند دوم : 11010 بعد از جهش بیت 2و3 ام : 10110 برازش : 8800 وزن :13
فرزند اول چون برازش بیشتری نسبت به والد دوم دارد پس آن را عوض میکنیم و فرزند دوم با برازش کمتر را عوض نمیکنیم . همچنین به برازش بهتری نرسیدیم پس میتوان ادامه داد الگوریتم را.

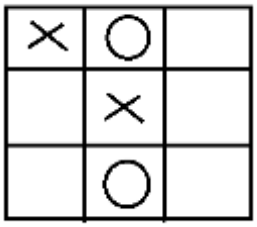
ب) جواب این سوال را به قطعیت نمیتوان گفت چون در کل برای این 5 بیت که هر کدام دو حالت دارند، پس 32 حالت متفاوت داریم وحتی در صورتی که تصور کنیم هر بار فرزندان تولید شده دو جایگشت متفاوت را نشان میدهند، پس در کل مینیمم 16 مرحله را نیاز داریم که همه حالت ها بررسی شوند.

.2

این سوال لازم است با الگوریتم ژنتیک حل شود که هر کروموزوم را یکی از خانه های خالی در ماتریس در نظر بگیریم که با حذف مقادیر ثابت 16 کروموزوم خواهیم داشت. همچنین باید یک مقدار احتمال بازترکیب و جهش داشته باشیم که مقدار کمی دارد و بین 0 و 1 است مثلا 0.1. در ادامه باید جمعیت اولیه را یعنی 16 کروموزوم را با اعداد رندوم می سازیم. سپس باید برازش جمعیت را بررسی کنیم که باید میزان تفاوت مجموع خانه های هر سطر، ستون، و قطر را با عدد 200 در نظر بگیریم. سپس با استفاده از بازترکیب و جهش فرزندان جدیدی تولید میکنیم که باید به صورت رندوم اتفاق بیفتد و با احتمال هایی که بالاتر گفتیم مقایسه شود و اگر کمتر بود انجام شود. روش بازترکیب به این صورت است که مثلا 2 سطر اول والد 1 با 3 سطر دوم والد 2 بازترکیب شده و فرزند بعدی هم 2 سطر اول والد 2 با 3 سطر دوم والد 1 بازترکیب میشود. حال همین مدل میتواند برای قطر و ستون هم انجام شود. برای جهش هم همین خانه هارا میتوان تغییرات کوچک به صورت رندوم داد. حال فرزندان تولید شده را برازششان را محاسبه میکنیم و در صورت بهتر بودن برازش نسبت به والدین در جمعیت اضافه میکنیم. شرط توقف الگوریتم هم زمانی است که مقدار برازش 0 شود.

.3

برای حل بازی دوز میتوان از الگورتیم مورچه ها استفاده کرد. در ابتدا میدانیم که هر استیت از بازی (هر جدول 3 در 3 که ممکن است خالی، پر یا بعضی از خانه هایش پر باشد) یک نود از گراف هستند. و هر حرکت از دو بازیکن یک یال از این نود از گراف به نود دیگری در نظر گرفته میشود. مورچه ها وقتی یک علامت در خانه ای از بازی قرار میدهد یا به سمت برد میرود یا به سمت باخت. و با ترشح فرومون نشان میدهند که چه مسیری بهتر است. اگر یک حرکت مثلا باعث بردن شود یا از بردن حریف جلوگیری کند، پس آن مسیر فرومون بیشتری دارد. حال برای حل این مسئله به همه ی یال های گراف یک مقدار رندوم و کم از فرومون خواهیم داد. سپس هر مورچه میتواند از وضعیتی که در آن قرار دارد حرکت کند و برای انتخاب حرکتش از یک تابع احتمالی با عامل اکتشافی و مقدار فرومون استفاده میکند. سپس بر اساس میزان خوب بودن فرومون ها را آپدیت میکند و مسیر هایی که به سمت موفقیت بوده اند فرومون بیشتری میگیرند. همچنین برای جلوگیری از همگرایی هم از مقدار فرومون ها به مرور کم میشود.



مثال : در اینجا مثلا اگر نوبت X باشد، مورچه در خانه های خالی میتواند حرکت کند. و حرکت به خانه (3,3) بیشترین فرومون را دارد چون باعث میشود که ببرد و احتمالا خانه های دیگر فرومون کمتری خواهند داشت.

.4

در این الگوریتم ژنتیک، هر کروموزوم نمایانگر یک مقدار کاندید برای ریشه‌های معادلات چندجمله‌ای داده شده است.

ابتدا جمعیت اولیه را تعریف میکنیم که شامل 100 تا عدد رندوم بین -100 تا 100 است.

```
def initialize_population():
    return np.random.uniform(low=-100, high=100, size=(population_size,))
```

درا دامه هزار بار جمعبت جدید تولید میکنیم تا به نزدیک ترین جواب برسیم . در هر generation، به ازای هر کروموزوم در جمعیت چک میکنیم که چقدر خطا داریم. این خطا به معنی میزان تفاوت مقدار معادله به ازای هر جواب، با ریشه ی مسئله که مقدار معادله را صفر میکند است. و در نهایت مقدار آن در $\frac{1}{1+error}$ قرار گرفته که هرچقدر ارور بیشتر برازش ما کمتر است.

```
def fitness(solution):
    error = 0
    for equation in equations:
        error += abs(equation(solution))
    return 1 / (1 + error)
```

بعد این برازش را برای اینکه بتوانیم درست مقایسه کنیم، تقسیم بر مقدار جمع برازش همه ی کروموزوم ها کرده و به یک مقدار نسبی میرسیم. حال برای تولید مثل، از این جمعیت با توجه به برازش بدست آمده 100 عضو انتخاب میکنیم.

```
probabilities = fitness_scores / np.sum(fitness_scores)
parent_indices = np.random.choice(range(population_size), size=population_size, p=probabilities)
```

با استفاده از والدین انتخاب شده عملیات بازترکیب و جهش را انجام میدهیم تا فرزندان جدید تولید شوند و آن هارا به جمعیت اضافه میکنیم. در عملیات بازترکیب از ترکیب خطی دو والدین برای ایجاد دو فرزند استفاده می کند و افراد جدیدی را به جمعیت معرفی می کند. در کد زیر alpha یک عدد رندوم بین 0 و 1 است.

```
def crossover(parent1, parent2):
    alpha = np.random.random()
    child1 = alpha * parent1 + (1 - alpha) * parent2
    child2 = alpha * parent2 + (1 - alpha) * parent1
    return child1, child2
```

در جهش برای حفظ تنوع ژنتیکی، تغییرات تصادفی را برای برخی افراد در جمعیت ایجاد می کنیم و از لوکال ماندن جواب جلوگیری میکنیم. در اینجا یک عدد رندوم تولید میکنیم و اگر از mutation rate کمتر بود یک عدد رندوم بین -1 تا 1 به کروموزوم اضافه میکنیم.

```
def mutate(solution):
    if np.random.random() < mutation_rate:
        mutation_amount = np.random.uniform(low=-1, high=1)
        solution += mutation_amount
    return solution
```

در اخر ریشه، کروموزومی که بیشترین مقدار fitness را دارد ریشه در نظر میگیریم. مقدار ریشه ی بدست آمده برای ورودی اول :

$2x - 4 = 0$

```
PS C:\Users\saba> & C:/Users/saba/anaconda3/python.exe "e
Roots found by genetic algorithm:
Equation: <function <lambda> at 0x000001F994D48AE0>
Root: 2.000413748972504
Value of the equation at the root: 0.0008274979450080266
```

$X^2 - 8x + 4 = 0$

```
PS C:\Users\saba> & C:/Users/saba/anaconda3/python.exe "e:/s
Roots found by genetic algorithm:
Equation: <function <lambda> at 0x00000249AAE48AE0>
Root: 0.5357963104146439
Value of the equation at the root: 0.0007072029367938271
```

$168x^3 - 7.22x^2 + 15.5x - 13.2 = 0$

```
PS C:\Users\saba> & C:/Users/saba/anaconda3/python.exe "e:/saba
Roots found by genetic algorithm:
Equation: <function <lambda> at 0x0000022FC4D48AE0>
Root: 0.3699271730639972
Value of the equation at the root: 0.050522349133679256
```

.5

در ابتدا پارامتر های مسئله را تعریف میکنیم. و ماتریس فرومون که مقدار فرومون های بین هر دو شهر را نشان میدهد را تعریف میکنیم.

```
def __init__(self, n_ants, n_iterations, alpha, beta, rho, Q):
    self.n_ants = n_ants
    self.n_iterations = n_iterations
    self.alpha = alpha
    self.beta = beta
    self.rho = rho
    self.Q = Q
```

```
def initialize(self, distances):
    self.all_distances = distances
    self.n_cities = len(distances)
    self.pheromone = np.ones((self.n_cities, self.n_cities)) / self.n_cities
```

در ادامه به ازای هر iteration همه ی مسیر ها و طولشان را محاسبه میکنیم. برای این کار به ازای هر مورچه ای که داریم یک مسیر که از یک شهر رندوم شروع میشود را میسازیم و تا زمانی که همه ی شهر هارا ببیند بر اساس مقدار فرومون و فاصله شهر ها مسیر را انتخاب میکند و این انتخاب بر اساس فرمول زیر انجام میشود.

$$P_{ij} = \frac{pheromone\ ij * 1/distance}{\sum_{all\ cities} heromone\ ij * 1/distance}$$

```
def probability(self, city_from, city_to, visited):
    pheromone = np.power(self.pheromone[city_from][city_to], self.alpha)
    distance = np.power(1.0 / self.distance(city_from, city_to), self.beta)
    return pheromone * distance
```

درواقع با توجه به مقدار احتمالی که بدست می آید، به صورت رندوم شهر بعدی را انتخاب میکند.

```
for ant in range(self.n_ants):
    route = []
    visited = set()
    current_city = np.random.randint(0, self.n_cities)
    route.append(current_city)
    visited.add(current_city)

    while len(visited) < self.n_cities:
        probabilities = []
        for city in range(self.n_cities):
            if city not in visited:
                probabilities.append(self.probability(current_city, city, visited))
            else:
                probabilities.append(0)

        probabilities = np.array(probabilities)
        probabilities = probabilities / probabilities.sum()
        next_city = np.random.choice(range(self.n_cities), p=probabilities)
        route.append(next_city)
        visited.add(next_city)
        current_city = next_city

    route.append(route[0])
    all_routes.append(route)
```

سپس مقدار فاصله ی مسیر بدست آمده را محاسبه میکنیم و اگر از مسیر قبلی بهتر بود آن را به عنوان پاسخ ذخیره میکنیم.

بعد از آن مقدار فرومون هارا آپدیت کرده که برای اینکار ابتدا مقادیر فرومون هارا در یک evaporation ضرب میکیم و این باعث کاهش مقدار همه ی فرومون ها میشود که از همگرایی زودرس جلوگیری شود. بعد هم با توجه به مقدار ثابت به روزرسانی فرومون و فاصله ی شهرها، و کیفیت مسیر، فرومون هارا به روزرسانی میکنیم.

```
def update_pheromones(self, all_routes, all_lengths):
    self.pheromone *= self.rho
    for i, route in enumerate(all_routes):
        for j in range(len(route) - 1):
            self.pheromone[route[j]][route[j + 1]] += self.Q / all_lengths[i]
```

نتیجه ی ورودی:

```
PS C:\Users\saba> & C:/Users/saba/anaconda3/pyth
Computentional intelligence/tamrin/2/5.py"
Best route: [0, 1, 3, 2, 0]
Best length: 80
```