1. 8-queen

세대 수 크기: 10

염색체 구성

: 염색체는 8x8 행렬에서 각 열에 하나의 퀸만 존재한다고 가정하고 각 행 인덱스를 값으로 갖는 정수 배열로 구성된다.

Ex) [0, 1, 2, 3, 4, 5, 6, 7] 염색체는 각각 [0][0], [1][1], [2][2], [3][3], [4][4], [5][5], [6][6], [7][7]에 존재하는 것이다.

적합도 함수

: 1열부터 7열까지 오른쪽에 있는 모든 열과 비교한다.

각 열에서 퀸이 위치하는 행을 j라고 가정한다.

오른쪽 열과 비교했을 때 오른쪽으로 n번째 있는 열에는 j행, j-n행, j+n행에 퀸이 존재하면 안 된다.

모든 퀸이 충돌하지 않을 때 그 쌍의 개수가 28이므로 적합도는 (28 – 충돌하는 쌍의 개수)가 된다.

Crossover 연산자

: 부모 염색체를 룰렛 휠 선택으로 뽑아 교차 지점을 랜덤으로 정해서 교차 연산을 한다.

Mutate 연산자

: 염색체의 각 유전자를 하나씩 탐색하며 돌연변이 확률에 걸린다면 행을 한 칸 증가시키도록 한다.

2. TSP

세대 수 크기: 6

염색체 구성

: 국내 주요 9개 도시를 중복없이 순회하는 TSP를 하기 위해서 다음과 같은 과정으로 염색체를 구성했습니다.

approach 1 : 서울에서 출발하여 서울로 도착하여야 합니다. 따라서, 서울을 제외한 8개 지역을 랜덤하게 선택하여 리스트에 담아주면 다음과 같이 구성됩니다.

리스트 1 = [춘천, 대전, 강릉, 대구, 부산, 광주, 울산, 인천]

리스트 2 = [부산, 춘천, 대구, 광주, 울산, 대전, 강릉, 인천]

두 개의 리스트가 있을 때 리스트를 중앙을 기준으로 교차하게 되면 다음과 같이 나옵니다.

리스트 1 = [춘천, 대전, 강릉, 대구, 울산, 대전, 강릉, 인천]

리스트 2 = [부산, 춘천, 대구, 광주, 부산, 광주, 울산, 인천]

교차를 통해 나온 두 개의 리스트를 보면 리스트 1은 대전, 강릉이 겹치게 됩니다. 도시의 이름을 유전자로 결정하게 되면 교차하는 과정에서 추가적인 연산이 필요하게 됩니다. 교차연산을 간소화하기 위해서 유전자의 배열을 다음 방법으로 변형해 주었습니다.

approach 2 : 우선 도시의 리스트를 만들어 주었습니다.
CITY = [서울, 인천, 대전, 춘천, 강릉, 대구, 울산, 부산, 광주]
approach 1에서 사용한 리스트 1와 리스트 2를 예시로 설명하겠습니다. 서울을 제외한 8개의 지역을 랜덤하게 선택하는 것은 동일합니다.
리스트 1 = [춘천, 대전, 강릉, 대구, 부산, 광주, 울산, 인천]
리스트 2 = [부산, 춘천, 대구, 광주, 울산, 대전, 강릉, 인천]
두 개의 리스트를 지역의 이름이 아닌 CITY 리스트에서 위치하는 인덱스로 바꾸어 주었습니다.
리스트 1의 경우 다음과 같은 과정을 거쳐 유전자를 생성합니다.
도시 리스트와 리스트 1에서 춘천의 위치를 비교합니다. 춘천은 CITY[1:]에서 2번째에 위치합니다.

CITY[1:] = [인천, 대전, 춘천, 강릉, 대구, 울산, 부산, 광주]
리스트 1 = [춘천, 대전, 강릉, 대구, 부산, 광주, 울산, 인천]

2. 리스트 1의 춘천을 2로 바꾸어 주고 CITY[1:]에 춘천을 삭제해 줍니다.

CITY[1:] = [인천, 대전, 강릉, 대구, 울산, 부산, 광주]
리스트 1 = [2, 대전, 강릉, 대구, 부산, 광주, 울산, 인천]

3. 리스트 1의 2번째는 대전입니다. CITY[1:0]에서 대전은 1번째에 위치합니다. 리스트 1의 대전을 1로 바꾸어주고 CITY[1:]에 대전을 삭제해 줍니다.

CITY[1:] = [인천, 강릉, 대구, 울산, 부산, 광주]
리스트 1 = [2, 1, 강릉, 대구, 부산, 광주, 울산, 인천]

4. 남은 강릉 대구 부산 광주 울산 인천도 다음과 같은 과정을 거칩니다.

CITY[1:] = [인천, 대구, 울산, 부산, 광주]
리스트 1 = [2, 1, 1, 대구, 부산, 광주, 울산, 인천]

CITY[1:] = [인천, 울산, 부산, 광주]
리스트 1 = [2, 1, 1, 1, 부산, 광주, 울산, 인천]

CITY[1:] = [인천, 울산, 광주]
리스트 1 = [2, 1, 1, 1, 2, 광주, 울산, 인천]

CITY[1:] = [인천, 울산]
리스트 1 = [2, 1, 1, 1, 2, 2, 울산, 인천]

CITY[1:] = [인천]
리스트 1 = [2, 1, 1, 1, 2, 2, 1, 인천]

CITY[1:] = []
리스트 1 = [2, 1, 1, 1, 2, 2, 1, 0]
이러한 과정을 거치면 리스트 1의 유전자가 얻어지게 됩니다. 리스트 2도 리스트 1와 같은 과정을 거치면 리스트 2 = [6, 2, 3, 4, 3, 1, 1, 0]를 얻을 수 있습니다.
리스트 1 = [2, 1, 1, 1, 2, 2, 1, 0]을 도시로 바꿔서 표현하는 방법은 다음과 같습니다.
리스트 1와 도시[1:]의 리스트를 비교합니다. 리스트 1의 맨 앞에는 2가 있습니다. CITY[1:]의 2번째는 춘천을 의미합니다.

리스트 1 = [2, 1, 1, 1, 2, 2, 1, 0]
CITY[1:] = [인천, 대전, 춘천, 강릉, 대구, 울산, 부산, 광주]

2. 춘천을 CITY[1:0]에서 삭제해주고 리스트 1의 2를 춘천으로 바꿔줍니다.

리스트 1 = [춘천, 1, 1, 1, 2, 2, 1, 0]
CITY[1:] = [인천, 대전, 강릉, 대구, 울산, 부산, 광주]

3. 리스트 1의 맨 앞에 1은 CITY[1:]에서 대전을 의미합니다. 리스트 1의 맨 앞 1을 대전으로 바꿔주고 CITY[1:]에서 대전을 삭제해줍니다.

리스트 1 = [춘천, 대전, 1, 1, 2, 2, 1, 0]
CITY[1:] = [인천, 강릉, 대구, 울산, 부산, 광주]

4. 위 와 같은 방법을 계속 진행하면 다음과 같은 결과를 얻을 수 있습니다.

리스트 1 = [춘천, 대전, 강릉, 1, 2, 2, 1, 0]
CITY[1:] = [인천, 대구, 울산, 부산, 광주]

리스트 1 = [춘천, 대전, 강릉, 대구, 2, 2, 1, 0]
CITY[1:] = [인천, 울산, 부산, 광주]

리스트 1 = [춘천, 대전, 강릉, 대구, 부산, 2, 1, 0]
CITY[1:] = [인천, 울산, 광주]

리스트 1 = [춘천, 대전, 강릉, 대구, 부산, 광주, 1, 0]
CITY[1:] = [인천, 울산]

리스트 1 = [춘천, 대전, 강릉, 대구, 부산, 광주, 울산, 0]
CITY[1:] = [인천]

리스트 1 = [춘천, 대전, 강릉, 대구, 부산, 광주, 울산, 인천]
CITY[1:] = []

위와 같이 리스트 1은 맨 처음 값을 넣은 리스트 1 = [춘천, 대전, 강릉, 대구, 부산, 광주, 울산, 인천] 과 동일한 도시를 가지게 됩니다. 숫자로 정의한 리스트 1와 리스트 2는 다음과 같습니다.

리스트 1 = [2, 1, 1, 1, 2, 2, 1, 0]
리스트 2 = [6, 2, 3, 4, 3, 1, 1, 0] 이 두 개의 리스트를 가운데를 기준으로 교차하면 다음과 같습니다.

리스트 1 = [2, 1, 1, 1, 3, 1, 1, 0]
리스트 2 = [6, 2, 3, 4, 2, 2, 1, 0]

리스트 1와 리스트 2를 위에서 설명한 도시로 바꾸는 과정을 거치면 다음과 같습니다.
리스트 1 = [춘천, 대전, 강릉, 대구, 광주, 울산, 부산, 인천]
리스트 2 = [부산, 춘천, 대구, 광주, 강릉, 울산, 대전, 인천]
따라서, 리스트 1와 리스트 2는 중복없이 교차 연산을 사용할 수 있게 됩니다.

적합도 함수
: 적합도 함수는 도시와 도시 사이의 거리를 2차원 리스트로 정의해주었습니다. 서울부터 시작하여 서울로 도착하기 때문에 숫자로 나열되어있는 유전자 리스트를 도시들의 이름으로 바꿔주었고, 맨 앞과 맨 뒤에 서울을 추가해 주었습니다. 리스트를 돌면서 도시와 도시 사이의

거리를 계산해 주었습니다. 모든 도시와 도시 사이의 거리가 1018이 되면 최소 거리라는 것을 이용하여 적합도가 1018이 되면 최적의 경로를 찾은 것으로 인지하였습니다.

crossover 함수

: 교차 함수는 다음과 같이 사용하였습니다. 개체 집단의 크기를 짝수로 해주었으며 population 리스트에 개체들이 들어가게 됩니다. 이때 population 리스트에서 적합도가 가장 작은 값이 개체들 중에서 가장 원하는 결과와 유사한 리스트라고 생각하여 개체들의 적합도를 오름차순으로 정렬해 주었습니다. 그다음 앞에서부터 적합도가 가장 낮은 개체 2개씩 꺼내서 교차하는 father와 mother로 사용해 주었습니다. 유전자를 교차시에는 유전자의 염색체가 8개이기 때문에 0부터 7까지의 숫자 중 랜덤하게 선택하여 반을 나누어서 결합해 주었습니다.

mutate 함수

: 돌연변이 함수를 설명하기 전에 우선 숫자로 가지는 유전자는 특징이 있습니다. 리스트의 위치마다 가질 수 있는 값의 최댓값이 정해져 있다는 점입니다. 유전자 [7, 6, 5, 4, 3, 2, 1] 8개의 염색체의 경우 유전자의 각 인덱스는 이와 같은 값을 최대로 가질 수 있는데, 그 이유는 CITY에서 해당하는 도시를 선택한 다음에는 지워줘야 하기 때문입니다. 따라서 유전자의 염색체는 0번째 index인 경우 0~7 사이 1번째 index인 경우 0~6 사이의 값을 가지도록 랜덤하게 설정해 주었습니다.

유전자 알고리즘 반복에 따른 적합도 함수의 추이 그래프

: max fitness - x축 : generation, y축 : 한 세대마다 가지는 최대 적합도 값

average fitness – x축 : generation y축 : 한 세대마다의 평균 적합도 값

최대 적합도 값은 population리스트의 0번째 염색체의 적합도 값으로 설정하고,

평균 적합도 값은 한 세대에 있는 모든 염색체들의 적합도 값을 더하고 이를 세대 수로 나누어 구한다.

모든 세대마다 y 값을 구하고, 실행 할 때마다 목표 적합도를 찾을 때까지의 모든 세대가 x축에 나오도록 함.

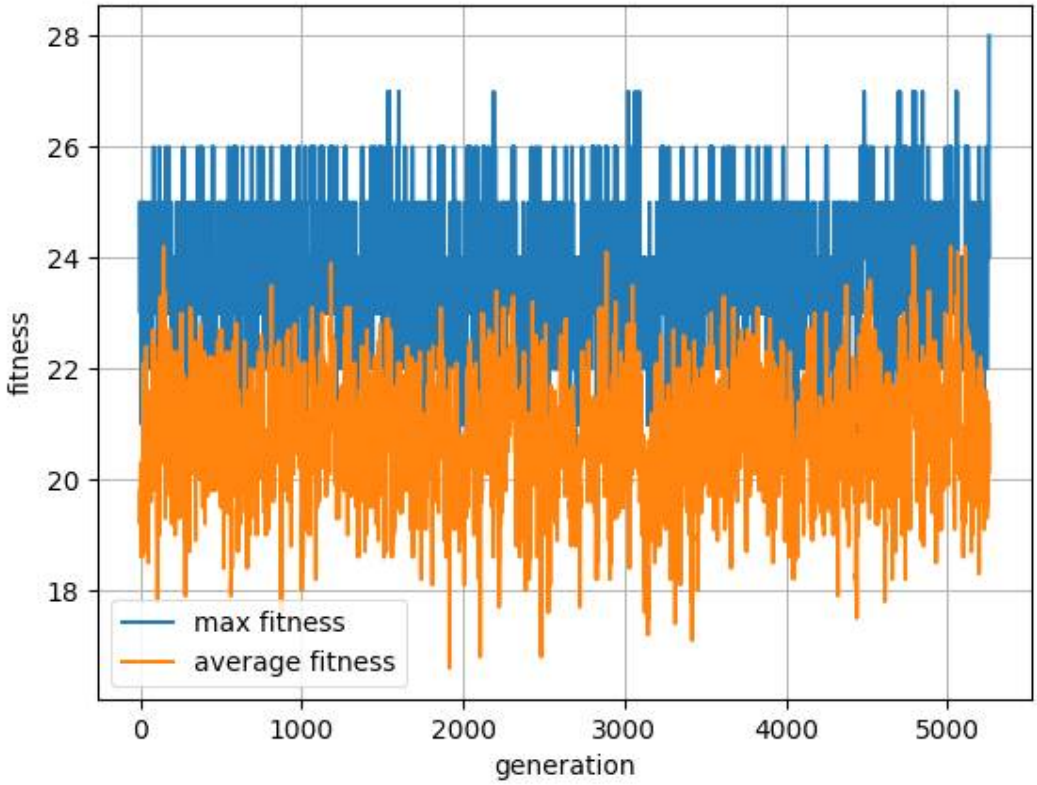
3. 토론 내용

김예훈 – 8queen 문제에서 염색체를 1차원 배열로 구성해 열 충돌을 막음. 적합도 함수에서 충돌하는 퀸을 찾는 조건문으로 오른쪽 열과 비교하는 3가지 조건을 가장 단순하면서 정확하게 구현하여 해당 코드를 팀 코드로 선정함.

윤승회 – TSP의 적합도 함수에서 도시와 도시 사이의 거리를 2차원 리스트로 정의하여 색다른 방법을 고안하고, 돌연변이 함수에서 리스트의 인덱스마다 가질 수 있는 값의 최댓값이 정해져 있다는 점을 고려해 구현함. 유저 친화적인 결과창을 추가함. 해당 코드를 팀 코드로 선정함.

차영원 – matplotlib을 이용해 세대에 따른 최대 적합도와 평균 적합도 두 가지의 값을 그래프로 깔끔하게 출력함. 실행할 때마다 종료될 때까지의 모든 세대가 x축을 이루도록 구현함. 해당 코드로 팀 소스의 추이 그래프를 출력함.

4. 팀 소스 실행 화면



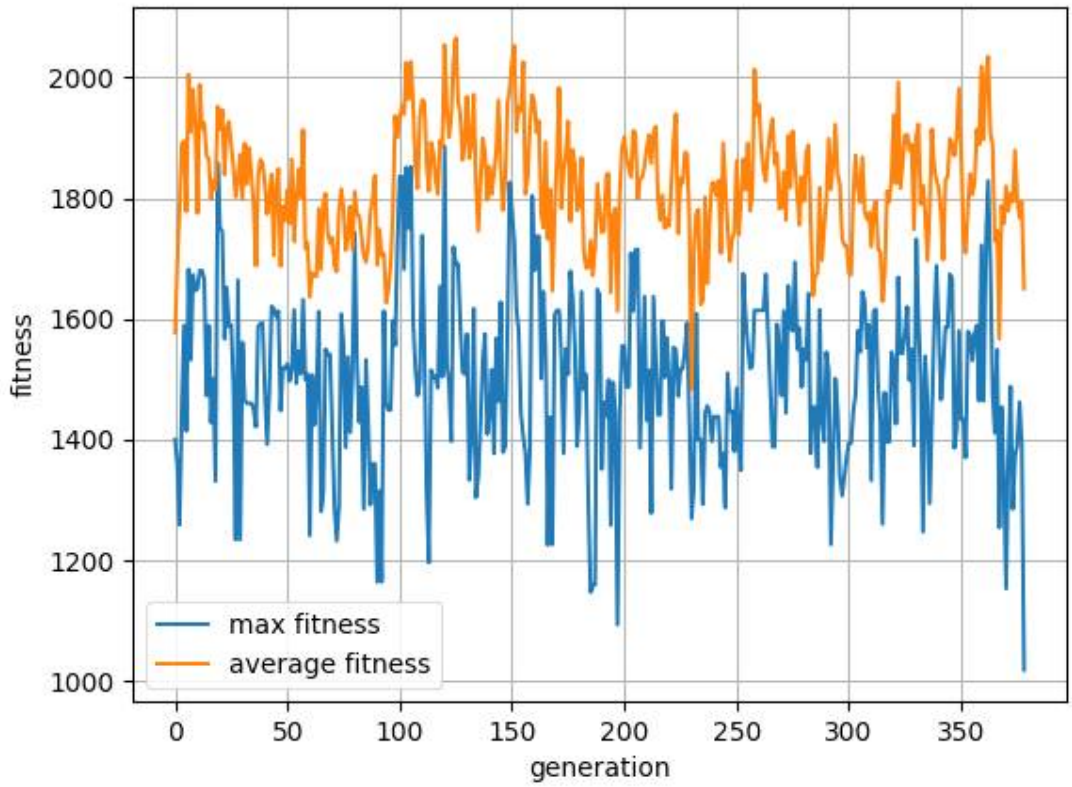

```
# 8x8 체스판에 8개의 퀸을 서로 잡아먹지 못하는 위치에 놓는다.
# 하나의 퀸을 기준으로 가로, 세로, 대각선에 다른 퀸이 존재하면 안된다.
import random
import matplotlib.pyplot as plt
import numpy as np
```

문제 출력 디버그 콘솔 터미널 JUPYTER Python Debug Console

```
세대 번호= 5259
염색체 # 0 = [3, 1, 6, 2, 5, 7, 4, 0] 적합도= 28
염색체 # 1 = [7, 6, 6, 0, 5, 7, 4, 1] 적합도= 24
염색체 # 2 = [6, 6, 5, 2, 5, 7, 4, 0] 적합도= 23
염색체 # 3 = [7, 5, 7, 1, 5, 7, 4, 0] 적합도= 21
염색체 # 4 = [7, 6, 6, 2, 5, 3, 6, 5] 적합도= 20
염색체 # 5 = [5, 6, 5, 2, 5, 7, 4, 0] 적합도= 20
염색체 # 6 = [7, 6, 6, 0, 5, 6, 4, 6] 적합도= 20
염색체 # 7 = [7, 6, 5, 6, 5, 7, 4, 1] 적합도= 20
염색체 # 8 = [7, 6, 7, 2, 5, 0, 6, 0] 적합도= 19
염색체 # 9 = [5, 5, 7, 2, 5, 7, 6, 5] 적합도= 15
```

							Q
	Q						
			Q				
						Q	
				Q			
		Q					
					Q		

8-queen 팀 소스 실행 화면과 추이 그래프 사진




```python
import random
import copy
import matplotlib.pyplot as plt

POPULATION_SIZE = 6     # 개체 집단의 크기
MUTATION_RATE = 0.1         # 돌연 변이 확률
SIZE = 8                # 하나의 염색체에서 유전자 개수
CITY = ["서울", "인천", "대전", "춘천", "강릉", "대구", "울산", "부산", "광주"]
DISTANCE = [[0,     30,     140,    75,     168,    237,    303,    325,    268],
            [30,    0,      140,    105,    198,    247,    315,    334,    257],
            [140,   140,    0,      173,    205,    119,    190,    200,    141],
            [75,    105,    173,    0,      102,    238,    293,    323,    313],
            [168,   198,    205,    102,    0,      213,    247,    287,    340],
            [237,   247,    119,    238,    213,    0,      71,     88,     173],
```

```
문제  출력  디버그 콘솔  터미널  JUPYTER                                Python Debug Console

염색체 # 5 = [4, 0, 5, 2, 1, 1, 0, 0] 적합도= 2191
염색체 # 5 = ['서울', '대구', '인천', '광주', '강릉', '춘천', '울산', '대전', '부산', '서울']

세대 번호= 378
염색체 # 0 = [0, 0, 5, 4, 3, 2, 1, 0] 적합도= 1018
염색체 # 0 = ['서울', '인천', '대전', '광주', '부산', '울산', '대구', '강릉', '춘천', '서울']
염색체 # 1 = [2, 1, 3, 3, 2, 2, 0, 0] 적합도= 1366
염색체 # 1 = ['서울', '춘천', '대전', '울산', '부산', '대구', '광주', '인천', '강릉', '서울']
염색체 # 2 = [6, 5, 3, 0, 0, 0, 0, 0] 적합도= 1560
염색체 # 2 = ['서울', '부산', '울산', '대전', '인천', '춘천', '강릉', '대구', '광주', '서울']
염색체 # 3 = [7, 1, 0, 4, 2, 1, 1, 0] 적합도= 1799
염색체 # 3 = ['서울', '광주', '대전', '인천', '부산', '대구', '강릉', '울산', '춘천', '서울']
염색체 # 4 = [1, 4, 1, 0, 1, 1, 0, 0] 적합도= 1958
염색체 # 4 = ['서울', '대전', '울산', '춘천', '인천', '대구', '부산', '강릉', '광주', '서울']
염색체 # 5 = [4, 0, 5, 2, 1, 2, 0, 0] 적합도= 2199
염색체 # 5 = ['서울', '대구', '인천', '광주', '강릉', '춘천', '부산', '대전', '울산', '서울']

PS C:\Users\user\Desktop>
```

TSP 팀 소스 실행 화면과 추이 그래프 사진