

3. 실습 2 - 외판원 순회 문제

1 문제 정의

외판원 순회 문제

1.

문제 정의

외판원 순회 문제는 시작 노드에서 출발해서 모든 노드를 방문하고 다시 시작 노드로 되돌아오는 최소 거리를 구하는 문제입니다.

데이터: code/06. 유전 알고리즘/외판원순회문제.csv

	S	A	B	C	D	E	F	G	H	I	J
S	0	14	17	23	21	18	17	3	25	15	22
A	14	0	2	3	9	16	18	27	1	22	6
B	17	2	0	6	20	10	15	23	5	28	10
C	23	3	6	0	27	12	12	3	29	23	1
D	21	9	20	27	0	15	5	19	22	29	11
E	18	16	10	12	15	0	12	15	15	20	10
F	17	18	15	12	5	12	0	14	24	9	8
G	3	27	23	3	19	15	14	0	27	15	26
H	25	1	5	29	22	15	24	27	0	9	24
I	15	22	28	23	29	20	9	15	9	0	28
J	22	6	10	1	11	10	8	26	24	28	0

- 시작점: S
- 지점: A ~ J
- 한 지점에서 같은 지점으로 가는 거리는 0
- 출발점부터 도착점까지 거리는 도착점부터 출발점까지 거리와 같음
- 따라서 여기서 사용하는 데이터는 대각 성분이 0인 대칭 행렬임

데이터

1. 문제 정의

데이터 불러오기

```
1 import pandas as pd
2 data = pd.read_csv("외판원순회문제.csv", index_col = 0)
3 display(data)
```

- 라인 2: 불러오고자 하는 데이터의 0번째 열이 인덱스로 사용돼야 하므로 index_col 인자를 0으로 설정했습니다. 이 인자는 인덱스로 사용할 칼럼을 설정합니다.

	S	A	B	C	D	E	F	G	H	I	J
S	0	14	17	23	21	18	17	3	25	15	22
A	14	0	2	3	9	16	18	27	1	22	6
B	17	2	0	6	20	10	15	23	5	28	10
C	23	3	6	0	27	12	12	3	29	23	1
D	21	9	20	27	0	15	5	19	22	29	11
E	18	16	10	12	15	0	12	15	15	20	10
F	17	18	15	12	5	12	0	14	24	9	8
G	3	27	23	3	19	15	14	0	27	15	26
H	25	1	5	29	22	15	24	27	0	9	24
I	15	22	28	23	29	20	9	15	9	0	28
J	22	6	10	1	11	10	8	26	24	28	0

거리 계산

1. 문제 정의

이 데이터에서 각 거리는 loc를 사용해 참조하겠습니다. 예를 들어, 경로 H> J> B의 거리는 H행 J열의 값과 J행 B열의 값을 더한 것으로 계산합니다.

거리 계산 예제

```
1 print(data.loc["H", "J"] + data.loc["J", "B"])
```

34

3. 실습 2 - 외판원 순회 문제

2 연산자 정의

해 표현 및 초기 해 집단 생성

유전 알고리즘의 해는 순열 인코딩을 사용해 표현하겠습니다.

유전자 예시

유전자 A

A	B	C	D	E	F	G	H	I	J
---	---	---	---	---	---	---	---	---	---

유전자 B

I	D	C	J	G	E	F	H	A	B
---	---	---	---	---	---	---	---	---	---

초기 해 생성

```
1 import numpy as np
2 def initialize(n, points):
3     X = [np.random.permutation(points) for _ in range(n)]
4     X = np.array(X)
5     return X
```

- 라인 3: points를 임의로 섞은 n 개의 배열로 구성된 X를 생성합니다. 즉, $X[i, j]$ 는 i번째 해가 시작점을 제외하고 j번째에 방문할 지점입니다.

적합도 함수

2.

연산자 정의

적합도는 거리가 가까울수록 높은 적합도를 갖도록 거리의 역수를 사용하겠습니다.

적합도 함수

```
1 def fitness(data, x):  
2     score = data.loc["S", x[0]]  
3     for i in range(len(x)-1):  
4         score += data.loc[x[i], x[i+1]]  
5     score += data.loc[x[-1], "S"]  
6     return 1/score
```

- 라인 2: "S"와 x[0]의 거리로 score를 초기화합니다.
- 라인 3~4: 모든 i에 대해 x[i]와 x[i+1]의 거리를 score에 더합니다.
- 라인 5: x의 마지막 요소와 "S"까지의 거리를 score에 더합니다.

선택 연산자

2.

연산자 정의

선택 연산자로는 룰렛 휠 방법을 사용하겠습니다.

선택 연산자

```
1 def selection(X, S, k):
2     selected_index = []
3     _S = S.copy()
4     for _ in range(k):
5         probs = _S / _S.sum()
6         x_idx = np.random.multinomial(1, probs).argmax()
7         selected_index.append(x_idx)
8         _S[x_idx] = 0
9     return X[selected_index]
```


교차 연산자

2. 연산자 정의

교차 연산자로는 순서가 있는 교차 연산자를 사용하겠습니다.

교차 연산자

```
1 def crossover(x1, x2):
2     start_idx = np.random.choice(range(0, len(x1)))
3     end_idx = np.random.choice(range(start_idx+1, len(x1) + 1))
4     new_x = np.empty(len(x1), dtype = object)
5     new_x[start_idx:end_idx] = x1[start_idx:end_idx]
6     new_x[~np.isin(x1, x1[start_idx:end_idx])] = x2[~np.isin(x2, x1[start_idx:end_idx])]
7     return new_x
```

- **라인 4:** 새로운 해를 빈 배열로 초기화합니다. 이때, 모든 해가 A부터 J까지의 문자열로 구성되므로 dtype을 object로 설정합니다. 만약 dtype을 설정하지 않으면 float 자료형으로 인식되어 라인 5와 6에서 값을 변경할 때 오류가 발생합니다.

돌연변이 연산자

2.

연산자 정의

돌연변이 연산자로는 순서 변경 돌연변이 연산자를 사용하겠습니다.

돌연변이 연산자

```
1 def mutation(x):  
2     a, b = np.random.choice(range(len(x)), 2, replace = False)  
3     (x[b], x[a]) = (x[a], x[b])  
4     return x
```

3. 실습 2 - 외판원 순회 문제

3 메인 함수

메인 함수

3.

메인 함수

```

1 def main(n, data, points, k, q, num_generation):
2     best_score = -1
3     X = initialize(n, points) # 초기해 생성
4     for _ in range(num_generation):
5         # 해 평가
6         S = np.array([fitness(data, x) for x in X])
7         current_best_score = S.max()
8         current_best_x = X[S.argmax()]
9
10        # 최고 해 업데이트
11        if current_best_score > best_score:
12            best_score = current_best_score
13            best_x = current_best_x
14
15        # k개 해 선택
16        X_new = selection(X, S, k)
17
18        # 교배 및 돌연변이 연산
19        children = []
20        for _ in range(n - k):
21            parent_idx = np.random.choice(range(k), 2, replace = False)
22            child = crossover(X_new[parent_idx[0]], X_new[parent_idx[1]])
23            if np.random.random() < q:
24                child = mutation(child)
25            X_new = np.vstack([X_new, child])
26        X = X_new
27    return best_x, best_score

```

- 라인 1: 세대당 해의 개수 n, 거리 행렬 data, 선택할 해의 개수 k, 돌연변이 확률 q, 세대 수 num_generation을 입력으로 받습니다.

외판원 순회 문제 해결

3. 메인 함수

유전 알고리즘의 하이퍼 파라미터 설정

```
1 n = 100
2 num_generation = 100
3 points = data.columns[1:]
4 k = 60
5 q = 0.05
```

- 라인 1~2: n을 100으로, num_generation을 100으로 설정함으로써 최대 1만 개의 해를 탐색합니다.
- 라인 3: data.columns의 0번째 요소는 S이므로 1번째 요소부터 슬라이싱합니다.

해 탐색

```
1 GA_best_x, GA_best_score = main(n, data, points, k, q, num_generation)
2 print(GA_best_x, 1/GA_best_score)
```

```
['G' 'C' 'A' 'H' 'B' 'J' 'E' 'D' 'F' 'I'] 79.0
```

랜덤 서치와 비교

3. 메인 함수

임의로 해를 만드는 함수인 initialize와 해를 평가하는 fitness 함수가 있으니, 이 두 함수를 활용해서 랜덤 서치를 구현하겠습니다.

랜덤 서치 구현

```
1 # 랜덤 서치
2 RS_best_score = -1
3 for x in initialize(10000, data.columns[1:]):
4     score = fitness(data, x)
5     if score > RS_best_score:
6         RS_best_score = score
7         RS_best_x = x
8
9 print(RS_best_x, 1/RS_best_score)
```

['F' 'D' 'J' 'E' 'B' 'C' 'A' 'H' 'I' 'G'] 90.0

- 유전 알고리즘을 사용해 찾은 해보다 좋지 않은 해를 찾았음
- 그러나 유전 알고리즘과 랜덤 서치 모두 임의성이 강하므로 실행할 때마다 결과에 차이가 있을 수 있음