AIP

HW 05 – REPORT

소속 :정보컴퓨터공학부

학번 : 202255535

이름 : 김진우

날짜 : 24.11.03

1. 서론

**Conventional Algorithm vs AI Algorithm**

Conventional Algorithm은 정확한 답을 찾을 수 있다는 장점이 있지만, TSP같은 복잡한 문제는 시간이 너무 오래 걸려 해결할 수 없는 경우가 생긴다는 단점이 있다. 이 문제를 해결할 수 있는 것이 AI Algorithm인데 이는 optimal solution을 보장하지는 못하지만 그에 근접한 답을 훨씬 빠른 시간 안에 찾아낼 수 있다는 장점을 가진다.

AI 알고리즘 중 가장 대표적인 것이 Hill Climbing인데 이번 과제에서 이와 관련해 numeric과 tsp문제를 풀어보며 이에 대해 학습할 수 있었다.

**Hill Climbing 알고리즘 (Steepest / First Choice )**

Hill-climbing 알고리즘은 목표 상태에 도달하기 위해 현재 상태에서 점진적으로 더 나은 상태로 이동하는 탐색 방법이다. 이 알고리즘에는 대표적으로 Steepest Ascent와 First Choice 방법이 있다.

- Steepest Ascent 방법: random하게 solution을 하나 만들고 현재 solution에서부터 현재 상태 주변의 모든 가능한 상태를 평가하여, 그중 가장 좋은 상태로 이동하는 방식이다. 이 방법은 모든 이웃 상태를 고려해 최선의 방향으로 진행한다는 특징이 있다.

- First Choice 방법: random하게 solution을 하나 만들고 현재 상태보다 더 나은 상태가 발견되면 즉시 그 상태로 이동하는 방식이다. 이는 steepest-Ascent와는 다르게 모든 이웃 상태를 살펴보지 않고 더 나은 상태를 찾을 때마다 이동하기 때문에 계산양이 줄어드는 장점이 있다.

**Numeric / TSP**

Numeric은 주어진 expression의 값을 가장 작게 하는 변수를 구하는 문제이고 TSP는 여러 도시를 모두 방문할 때 최소한의 cost가 들게 하는 순서를 구하는 문제이다.

2. 본론

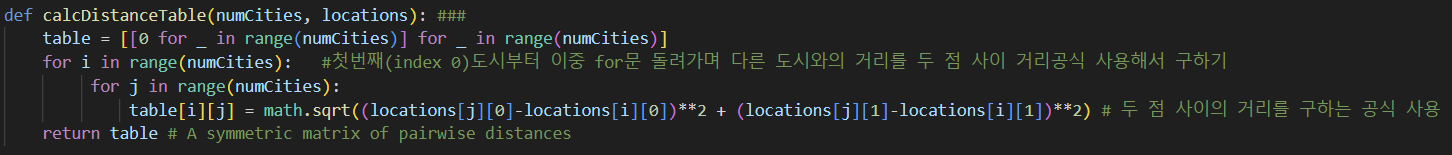
**2.1 tsp.py**

tsp.py에서는 steepest ascent(tsp)와 first-choice(tsp)를 풀 때 공통으로 사용되는 함수와 global 변수 와 같은 것들이 선언되어 있다. NumEval은 evaluation의 수를 의미하는데 여기서는 evaluate함수 호출 횟수를 의미한다.

주요 함수들을 하나하나 살펴보면 아래와 같다.

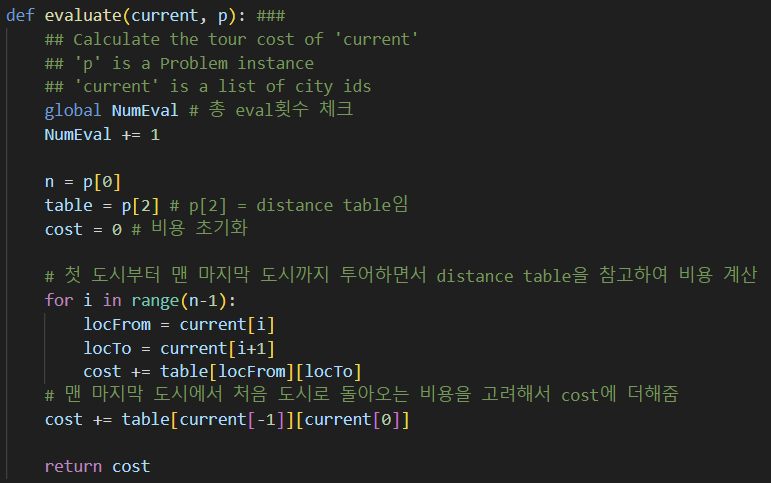
- **createProblem()** : txt파일을 읽어와서 문제를 만드는 함수이다.

- **calcDistanceTable(numCities, locations)** : 도시 간의 거리를 계산해서 이를 2차원 matrix으로 만들어준다. 이때 도시 사이의 거리는 거리공식을 사용해 구한다.



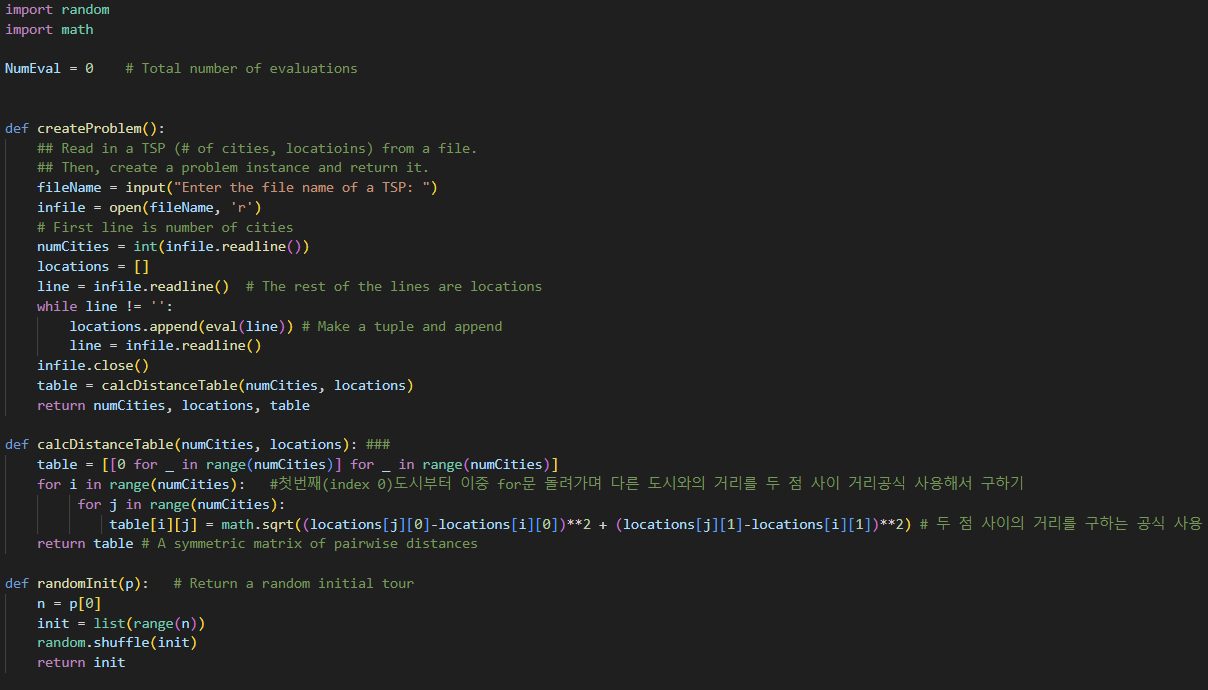
- **randomInit(p)** : random하게 초기 solution을 하나 만들어준다.

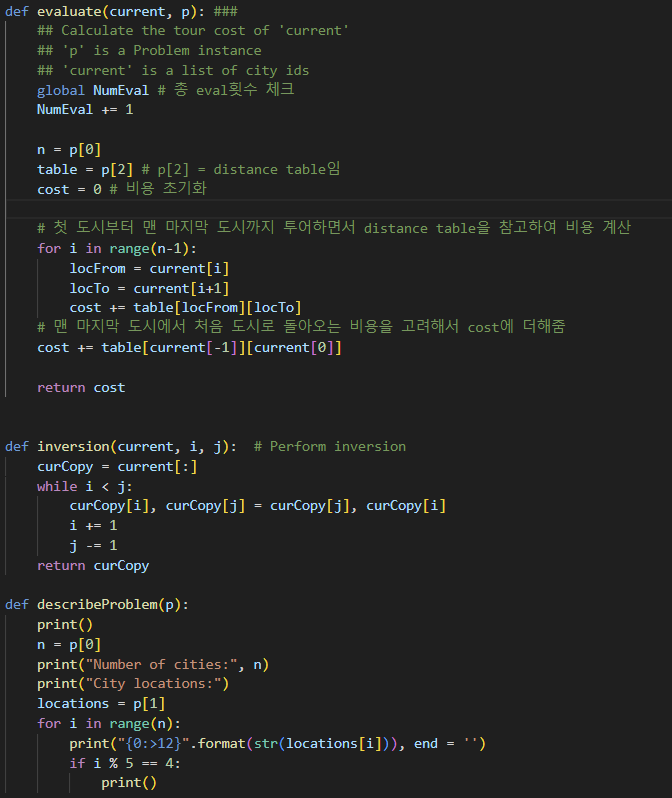
- **evaluate(current, p)** : solution의 cost를 계산하는 함수이다. 이 함수가 실행되면 current의 순서대로 cost가 계산되게 된다. 이때 이전에 만들어둔 2차원 matrix인 table을 사용하게 되는데 table[i][i+1]을 계속해서 더하다 보면 전체 cost를 구할 수 있게 된다.

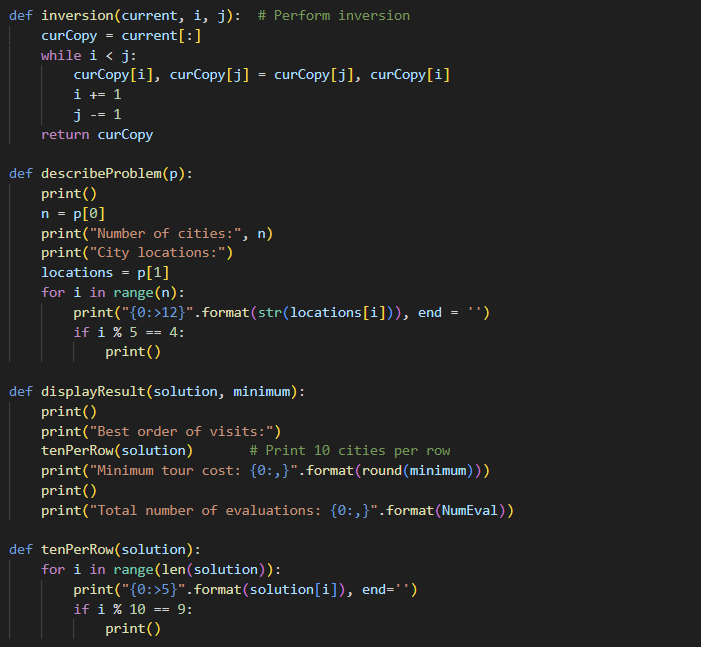


- **inversion(current, i, j)** : 도시의 방문위치를 바꿔준다.

[전체 코드]





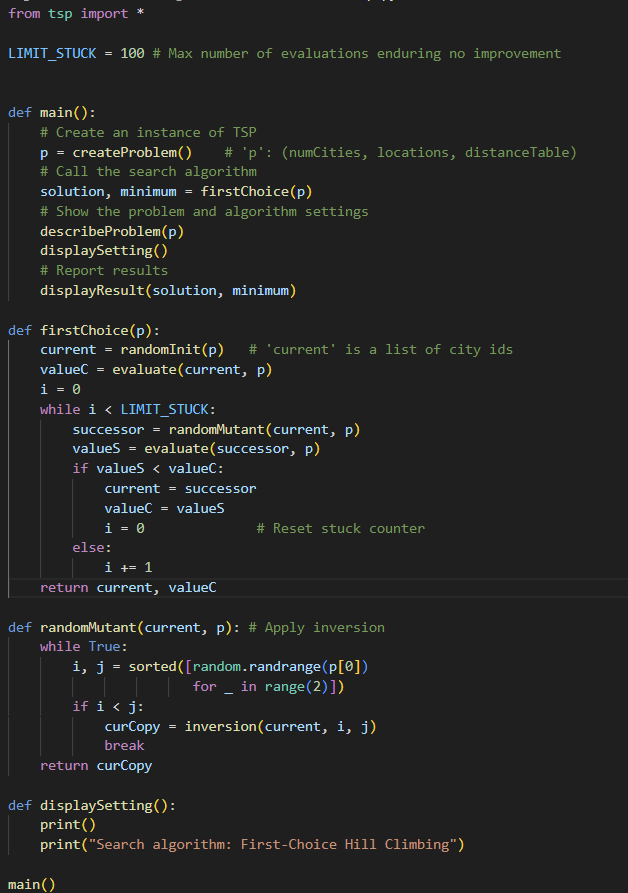


**2.1.1. first-choice (tsp)**

First-choice(tsp).py는 tsp.py를 import한다.

여기서 중요한 함수는 first-choice와 randomMutant함수이다.

[전체 코드]

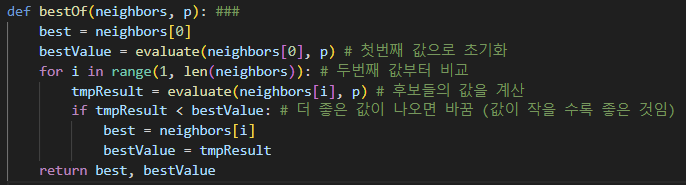


**2.1.2. steepest ascent (tsp)**

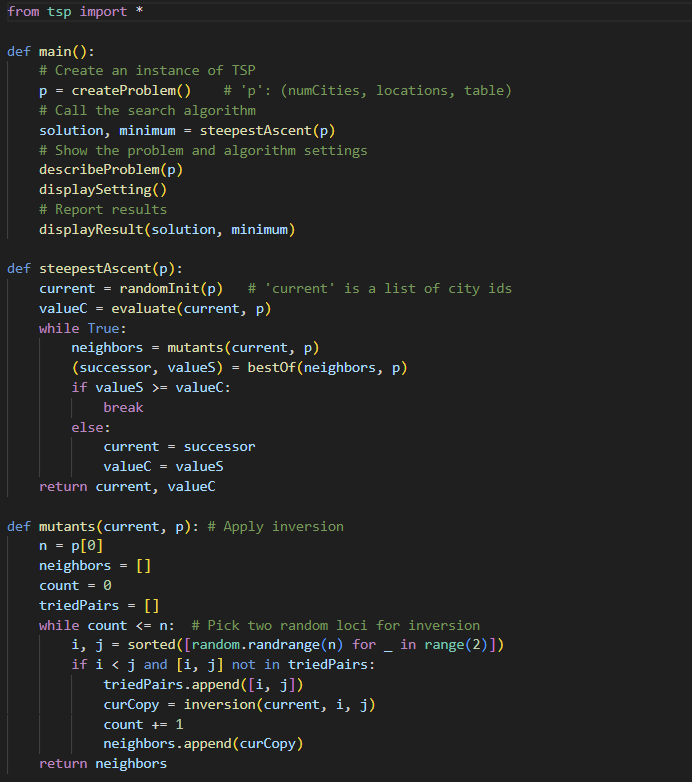
Steepest ascent (tsp).py 역시 tsp.py를 import한다.

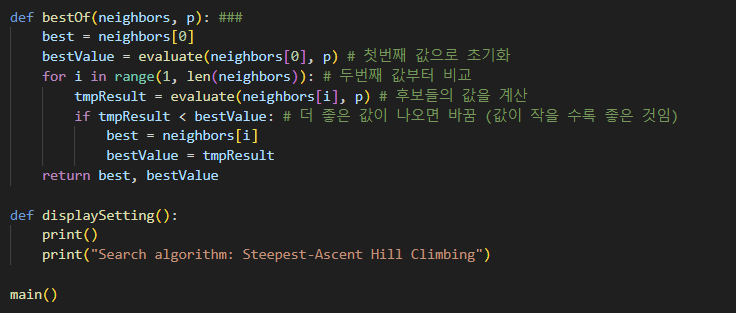
여기서 구현해야 하는 주요 함수는 bestOf(neighbors,p)이다.

- **bestOf(neighbors, p**) : 이 함수는 가장 좋은 값을 가지는 neighbor를 리턴하는 함수인데 값이 좋다는 것은 value가 더 작은 상황을 의미한다. 이때 여기서는 tsp.py의 evaluate 함수를 사용한다.



[전체 코드]



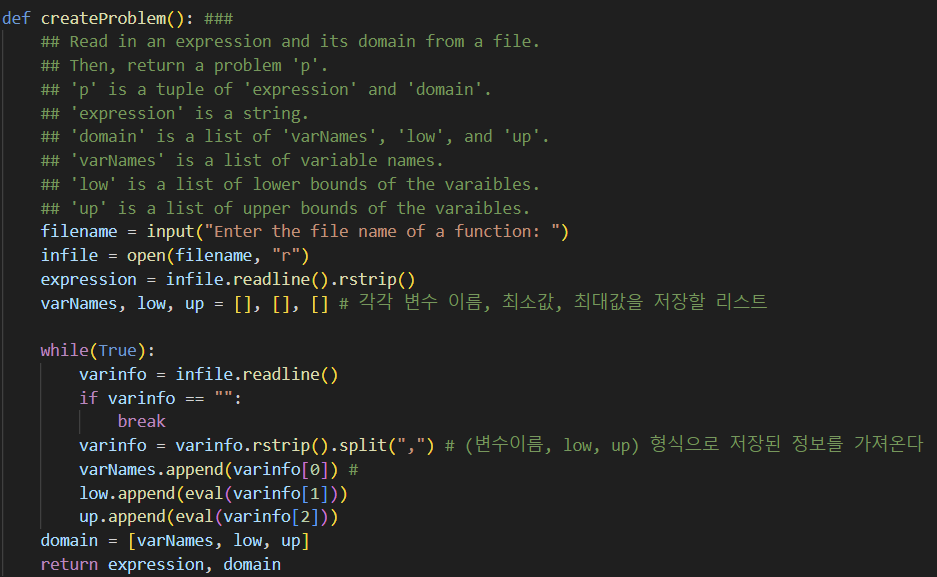


**2.2 Numeric.py**

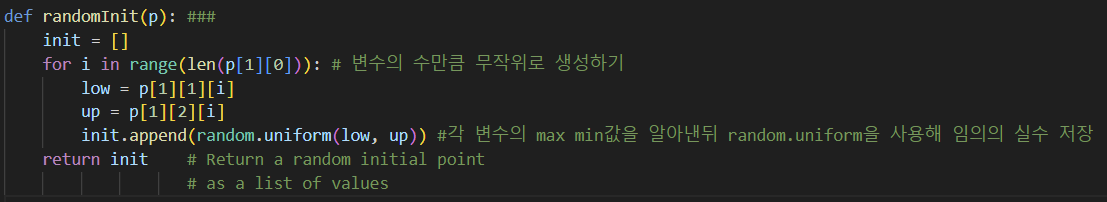
Numeric.py에는 steepest ascent(n)과 first-choice(n)을 풀 때 공통으로 사용되는 함수와 global 변수 와 같은 것들이 선언되어 있다. DELTA값은 step-size를 의미하고, NumEval은 evaluation의 수를 의미하는데 여기서는 evaluate함수 호출 횟수를 의미한다.

함수 하나하나를 따져보면 아래와 같다.

- **createProblem()** : txt파일을 읽어와서 이를 return 해준다. 이때 return 값인 expression은 함수식이고, domain은 varNames, low, up으로 구성된 list로 각각은 변수이름, 상한값, 하한값을 의미한다.



- **randomInit(p)** : 변수들의 값을 초기화해주는 함수로 random.uniform을 통해 범위 안의 랜덤한 x 값을 가져온다.



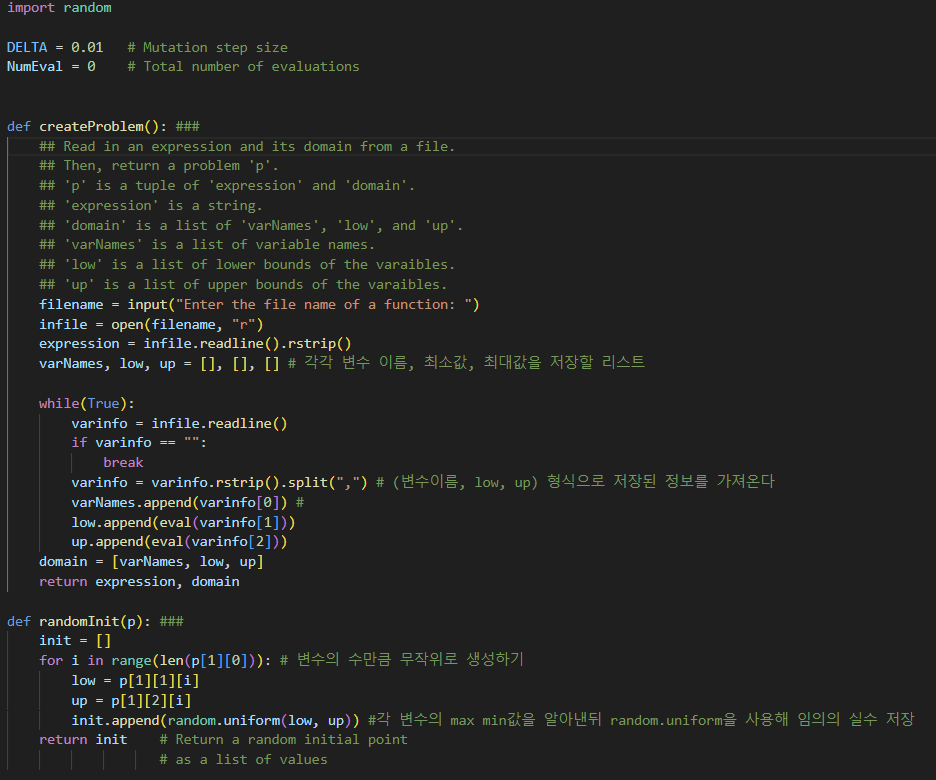
- **evaluate(current, p)** : 변수의 값을 통해 이를 evaluate하고 NumEval을 계산

- **mutate(current, i, d, p)** : 현재 값으로부터 값을 변형

- **describeProblem(p)** : 읽어온 파일을 출력

- **displayResult(solution, minimum) :** solution을 터미널에 출력

[전체 코드]

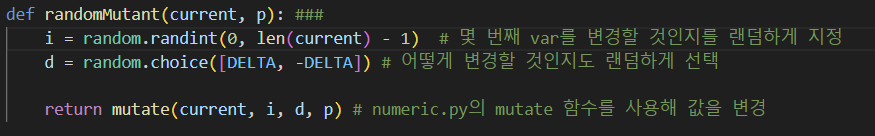




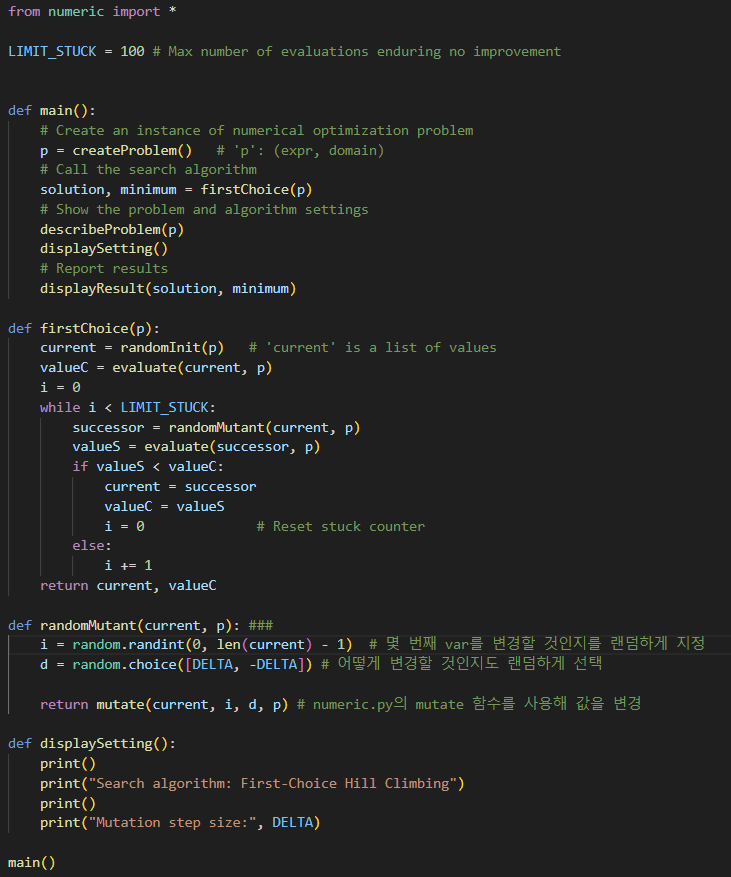
**2.2.1. first-choice (n)**

First-choice(n).py는 numeric.py를 import한다. 여기서 LIMIT\_STUCK이라는 것이 사용되는데 이는 Max number of evaluations enduring no improvement를 뜻한다. 즉 LIMIT\_STUCK의 수만큼 randomMutant함수를 실행하였음에도 불구하고 값이 improve되지 않았다면 종료되게 되는 것이다. 여기서 구현해야 하는 주요 함수는 randomMutant함수이다.

- **randomMutant(current, p**) : 이 함수를 통해 랜덤한 i번째 변수에 d만큼의 변형을 준다. 이때도 역시 numeric.py의 mutate함수를 통해 값을 변경하고 d는 문제에서 지정해준 대로 +DELTA/-DELTA중 하나로 선택된다.



[전체 코드]

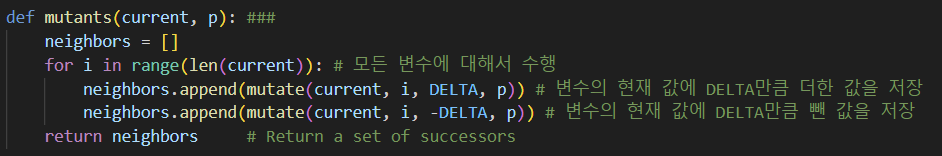


**2.2.2. steepest ascent (n)**

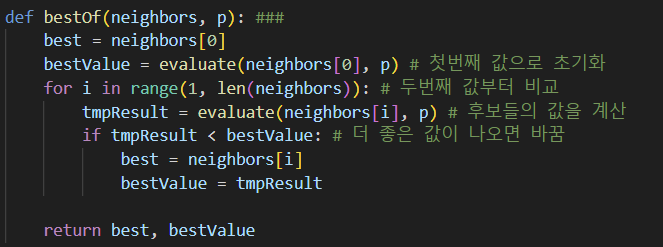
steepest ascent (n).py 역시 구현해둔 numeric.py를 import한다.

여기서 구현해야 하는 주요 함수는 mutants와 bestOf함수이다.

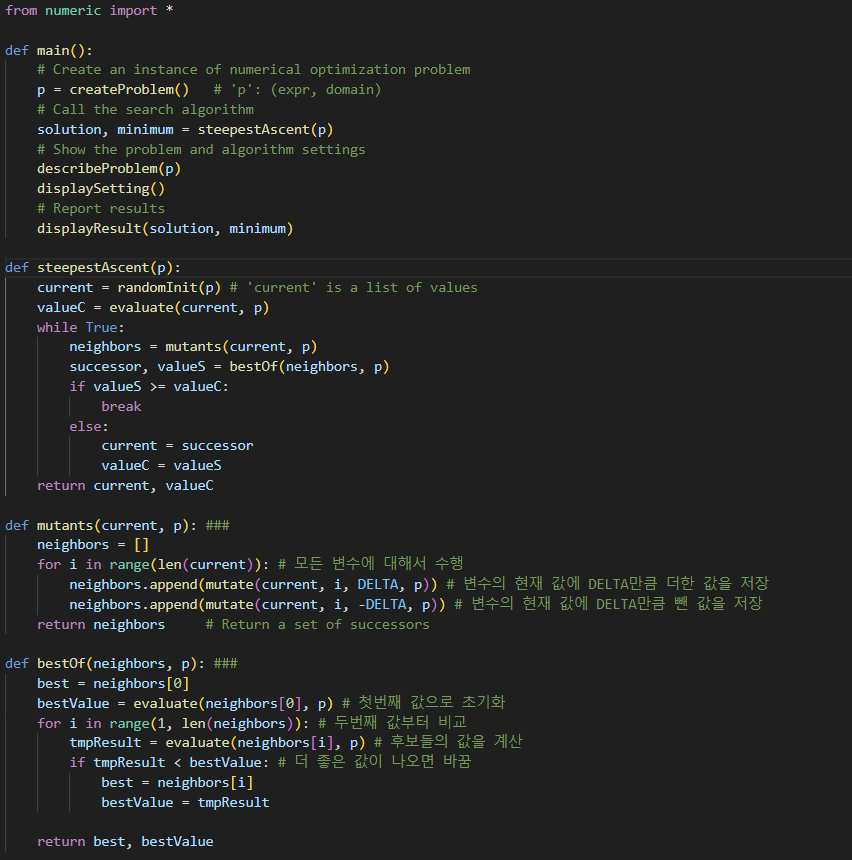
- **mutants(current, p)** : 이 함수는 변수의 현재 값에 DELTA만큼 더하거나 뺸 값으로 neighbors를 만든다. 이때 numeric.py에 선언해둔 mutate함수가 사용되는데 current는 현재 값, i는 몇 번째 변수, d는 변형되는 정도, p는 주어진 변수들을 의미한다. mutate함수는 low, up 범위를 넘어서면 원래의 값을 그대로 리턴하기 때문에 범위를 벗어나지 않는 neighbors들만 저장되게 된다.

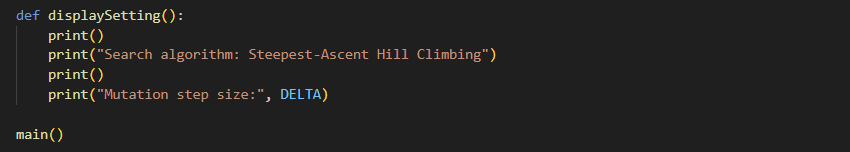


- **bestOf(neighbors, p)** : 이 함수는 mutants를 통해 얻은 이웃들 중 가장 좋은 이웃을 찾는 함수이다. 이때 numeric.py의 evaluate함수를 사용한다.



[전체 코드]



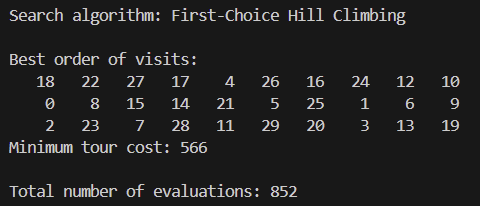


**2.3 결과 화면**

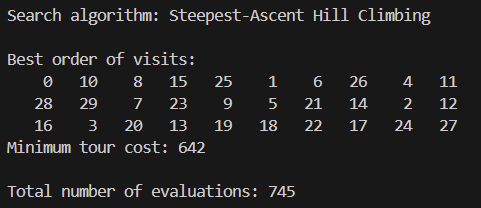
**2.3.1. TSP문제** - First-choice/ Steepest Ascent

**1) problem/tsp30.txt**

[first-choice]

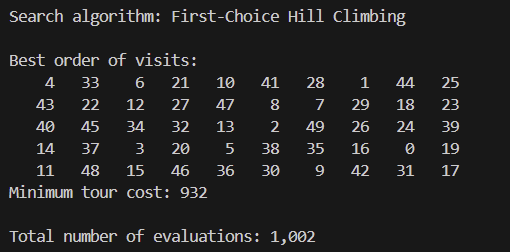


[Steepest Ascent]

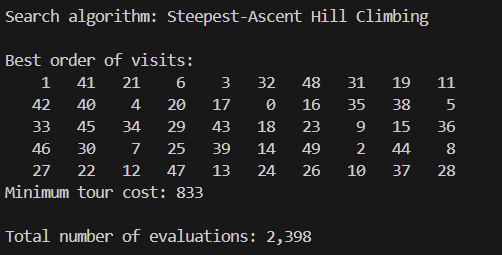


**2) problem/tsp50.txt**

[first-choice]

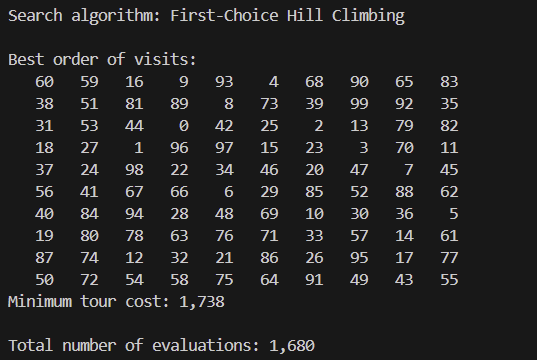


[Steepest Ascent]

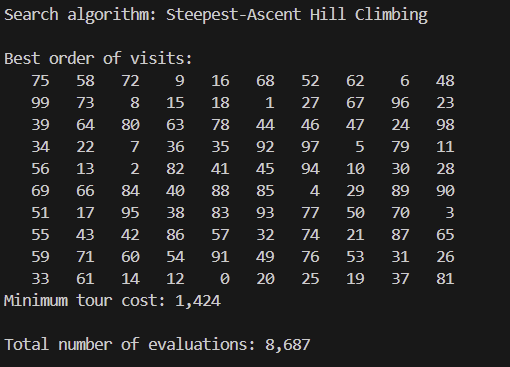


**3) problem/tsp100.txt**

[first-choice]



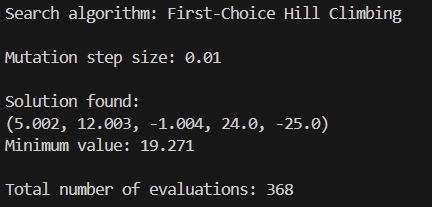
[Steepest Ascent]



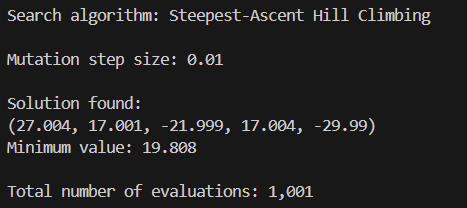
**2.3.2. Numeric 문제** - First-choice/ Steepest Ascent

**1) problem/Ackely.txt**

[First-choice]

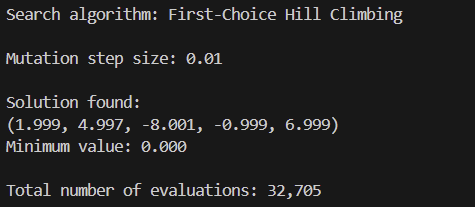


[Steepest ascent]

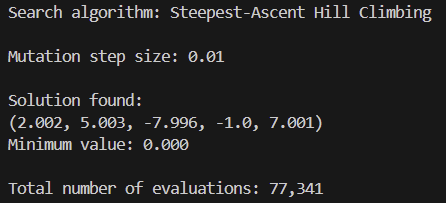


**2) problem/Convex.txt**

[First-choice]

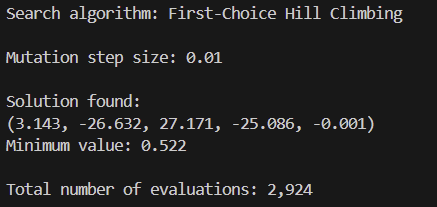


[Steepest ascent]

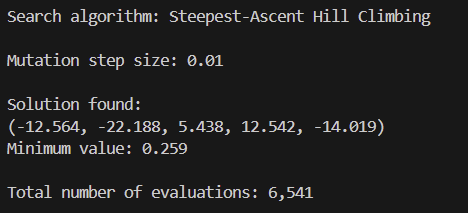


**3) problem/Griewank.txt**

[First-choice]



[Steepest ascent]



3. 결론

<2.3>의 결과를 보면 tsp문제와 numeric문제 모두 first-choice와 steepest ascent 방법이 비슷한 solution을 찾아내지만, evaluation의 횟수가 first-choice방법이 steepest ascent 방법보다 훨씬 적다는 것을 확인할 수 있다. 이는 서론에서 말했듯 first-choice 방법은 steepest-Ascent와는 다르게 모든 이웃 상태를 살펴보지 않고 더 나은 상태를 찾을 때마다 이동하기 때문이라고 생각할 수 있다.

또한 코드를 돌릴 때마다 결과가 계속 다르게 나오고, 가끔씩 다른 답들과 많이 다른 결과가 나오기도 했다. 이는 아마도 알고리즘의 특성상 가장 가까운 상태들만 확인하기 때문에 local minimum에 빠지는 등의 경우로 인해 발생한 결과라고 볼 수 있었다. 그럼에도 불구하고 많은 경우에 서로 유사한 결과가 나옴을 확인할 수 있었고, Conventional Algorithm으로 시간 내에 해결할 수 없는 문제를 AI Algorithm으로 해결할 수 있는 새로운 방법을 알게 되었다.