TSP Project

۱)

تابع nearest neighbor:

ابتدا تمام نقطه‌ها، ملاقات‌نشده فرض شده‌اند. (not_visited)

نقطه اول به مسیر(path) اضافه می‌شود و از not_visited حذف می‌شود.

تا زمانیکه not_visited خالی نشده باشد، برای آخرین نقطه‌ای که در مسیر وجود دارد، فاصله‌اش با نقاطی که در not_visited هستند محاسبه می‌شود و نقطه‌ای که کمترین فاصله با آن را دارد به عنوان نزدیکترین همسایه ذخیره می‌کنیم.

نزدیکترین همسایه وارد path می شود و از not_visited حذف می‌شود.

تا زمانیکه نقاط ملاقات‌نشده داریم، این کار را ادامه می‌دهیم.

تابع exhaustive(permutation):

تمام جایگشت‌های نقاط با استفاده از دستور permutation از کتابخانه itertools در یک لیست گذاشته می‌شود.(بعد از اینکه جایگشت‌های نقاط رو پیدا کردیم،نقطه شروع را به اول و آخر لیست اضافه می‌کنیم.)

با صدا کردن تابع calculate_distances روی هر permutation در لیست، آن جایگشتی که کمترین طول مسیر را دارد پیدا می‌کنیم.

تابع distance_between_two_points:

این تابع فاصله بین دو نقطه (x,y) و (a,b) را با توجه به رابطه زیر بدست می‌آورد:

$$d = \sqrt{(x-a)^2 + (y-b)^2}$$

تابع calculate_distances:

لیستی از نقاط دریافت می‌کند و مجموع فواصل بین نقاط را با استفاده از تابع distance_between_two_points بدست می‌آورد.

۲)

مرتبه زمانی nearest neighbor:

```python
def nearest_neighbor_algorithm(points):
    path = []
    not_visited = points
    path.append(points[0])
    not_visited.remove(points[0])

    while len(not_visited) > 0:
        not_visited = sorted(not_visited, key=lambda k: distance_between_two_points(path[-1], k))
        nearest_neighbor = not_visited[0]
        path.append(nearest_neighbor)
        not_visited.remove(nearest_neighbor)
    return path
```

در بدترین حالت تمام نقاط(به جز نقطه اول) در not_visited هستند. پس حلقه‌ی while، از مرتبه n است.
خط ۸: ابتدا n-2 همسایه را بررسی می‌کند، در دور بعد n-3 همسایه و ... تا هیچ همسایه‌ای باقی نماند، پس از مرتبه‌ی n است.

$$time\ complexity = o(n \times n) = o(n^2)$$

مرتبه زمانی exhaustive(permutation):

```python
def exhaustive_algorithm(points):
    perm = []
    for permutation in permutations(points):
        permutation = list(permutation)
        first = permutation[0]
        permutation.append(first)
        perm.append(permutation)

    perm = sorted(perm, key=calculate_distances)

    # path=perm[0]
    return perm[0]
```

چون تمام جایگشت‌های n نقطه باید محاسبه شوند حلقه for از مرتبه !n است.
در خط ۸ به ازای هر permutation، تابع calculate_distance صدا زده می‌شود و مسافت n نقطه را محاسبه می‌کند، پس از مرتبه‌ی n!n است.

$$time\ complexity = o(n! \times n)$$

۳)

n=5 , permutation	n=5 , nearest neighbor
0.00056200000	0.00002300000
0.00055700000	0.00002200000
0.00049100000	0.00002000000
average= 0.000536666666667	average= 0.000021666666667

n=9 , permutation	n=9 , nearest neighbor
2.65178600000	0.00005800000
2.55848000000	0.00006100000
2.57089500000	0.00005900000
average= 2.593720333333333	average= 0.000059333333333

n=8 , permutation	n=8 , nearest neighbor
0.26922800000	0.00004500000
0.25407200000	0.00006700000
0.27040800000	0.00004500000
average= 0.264569333333333	average= 0.000052333333333

n=10 , permutation	n=10 , nearest neighbor
30.11067100000	0.00007000000
28.17740900000	0.00006100000
29.96211100000	0.00006400000
average= 29.416730333333333	average= 0.000065

۴)

حالت n=10 و n=8 را درنظر میگیریم:

حالت nearest neighbor:

$$T = o(n^2)$$

$$\frac{10^2}{8^2} = 1.5625 \qquad \frac{0.000065}{0.00005233333} = 1.2420382165684 21$$

پس حالت تئوری و واقعی تقریبا برابرند.

حالت permutation:

$$T = o(n! \times n)$$

$$\frac{10! \times 10}{8! \times 8} = 112.5 \qquad \frac{29.416730333333333}{0.264569333333333} = 111.187226284729536$$

پس حالت تئوری و واقعی تقریبا برابرند.