Mehmet Nesih Kardaş
22080410020

Algoritma Analizi-Ödev 4

```
1. Şehirleri bir dizi olarak temsil edin (örneğin, cities = [0, 1, 2, ..., n-1]).
2. Başlangıç şehri seçin (örneğin, current_city = 0).
3. Ziyaret edilen şehirleri bir küme olarak saklayın (örneğin, visited = {0}).
4. Turu bir liste olarak başlatın (örneğin, tour = [0]).
5. Döngü (len(visited) < n):
    6. En yakın ziyaret edilmemiş şehri bulun (örneğin, next_city = find_nearest_unvisited_city(current_city, visited, cities)).
    7. next_city'yi ziyaret edin (örneğin, visited.add(next_city)).
    8. Turu güncelleyin (örneğin, tour.append(next_city)).
    9. current_city'yi next_city olarak ayarlayın (örneğin, current_city = next_city).
10. Başlangıç şehrine dönün (örneğin, tour.append(tour[0])).

11. Optimal TSP turunu yazdırın:
    for city in tour:
        print(city)
```

Algoritmanın Analizi

1.İteratif Algoritma:,

Karmaşıklık Analizi

- O(n!), n şehir sayısını temsil eder. Bu, algoritmanın şehirlerin sayısının artmasıyla birlikte faktöriyel olarak yavaşladığını gösterir. Örneğin, 10 şehir için 3,628,800 farklı rota hesaplanmalıdır.
- O(n), algoritmanın her zaman için tuttuğu geçici rota listesi ve en iyi rota listesi dışında önemli bir ekstra hafıza kullanımı yoktur. Yani, algoritma girdi büyüklüğüne doğrusal olarak ölçeklenir.

2.Özyinelemeli Algoritma:

Karmaşıklık Analizi

- O(n!), çünkü bu yöntem de tüm olası rotaları denemektedir. Her seviyede, kalan şehirlerin sayısına eşit sayıda yinelemeli çağrı yapılır. Bu da faktöriyel bir büyüme demektir
- O(n), bu yaklaşım için de geçerlidir, çünkü her çağrıda rotanın şu anki durumu ve kalan şehirler için yeni listeler oluşturulur. Ancak, yinelemeli çağrılar nedeniyle çağrı yığını kullanımı burada daha belirgindir.
- Her yinelemeli çağrı, çağrı yığınında ekstra bir alan kaplar, bu da derinliği n'ye kadar olan bir yığın kullanımına yol açar.