ALGORİTMA ANALİZİ-ÖDEV 4

Kaba Kod

1. Başla
2. Mesafeler matrisini oluştur: İlgili şehirler arası mesafeleri içeren bir matris oluşturulur.
3. Başlangıç şehri belirle: Gezgin satıcının başlayacağı şehir seçilir.
4. Tüm şehirleri ziyaret etme sırasını belirle: Başlangıç şehri hariç tüm şehirler bir listeye eklenir.
5. Gezgin Satıcı Problemini Çöz: TSP fonksiyonu çağrılarak en kısa yolun ağırlığı hesaplanır.
6. Sonucu ekrana yazdır.
7. Bitir

ALGORİTMANIN ANALİZİ

1. Özyinelemeli (Recursive) Algoritma

```
  visited[currentCity] = 1;                                                    // 1

  minPath[depth] = currentCity;                                                //1

  if (depth == numCities - 1)                                                 // n-1
  {
     if (graph[currentCity, 0] != 0 && cost + graph[currentCity, 0] < minCost)   // 1
     {
        minCost = cost + graph[currentCity, 0];                                  // n+1
        minPath[numCities] = 0;                                                 // 1
        Array.Copy(minPath, 0, minPath, 0, numCities + 1);                      // n
     }
  }
  else // O(n)
  {
     for (int i = 0; i < numCities; i++)                                      // 2n+1
     {
        if (visited[i] == 0 && graph[currentCity, i] != 0)                     // 1
        {
           TSP(i, cost + graph[currentCity, i], depth + 1);                   // (n-1)!
        }
     }
  }

  visited[currentCity] = 0;                                                    // 1
}
                                                                              //Sonuc= n * (n-1)!
```

- Zaman Karmaşıklığı (Time Complexity):
    - Bu iteratif çözümde, tüm olası şehir kombinasyonlarını kontrol etmek için bir dış döngü ve her bir kombinasyonun maliyetini hesaplamak için iç içe döngüler kullanılır.

- Dış döngü, tüm şehir kombinasyonlarını kontrol ederken iç döngü, her bir kombinasyonun maliyetini hesaplar.
- Dış döngünün karmaşıklığı O((n-1)!) ve iç döngünün karmaşıklığı O(n) şeklinde ifade edilebilir. Burada n, şehir sayısıdır.
- Dolayısıyla, iteratif çözümün zaman karmaşıklığı O((n-1)! * n) olacaktır.

- Bellek Karmaşıklığı (Space Complexity):

  - Matris dışında sadece birkaç ekstra değişken kullanılır.
  - Ekstra değişkenlerin bellek karmaşıklığı sabit olduğundan, iteratif çözümün bellek karmaşıklığı O(1) olacaktır.

2. Yinelemeli (Iteratif) Algoritma

```
    if (level == cities && AllVisited())                                    // n
    {
        return ComputePath(path);                                           // n
    }

    int minPath = int.MaxValue;                                             // 1

    for (int i = 0; i < cities; i++)                                        // 1+ n-1 +n=2n
    {
        if (visited[i] == 0) // 1
        {
            visited[i] = 1; // 1
            path[level] = i; // 1
            minPath = Math.Min(minPath, TSP(path, level + 1));              // n!
            visited[i] = 0;                                                 // 1
        }
    }

    return minPath;                                                          // 1
}                                                                           //Sonuç= O(n!)
```

- Zaman Karmaşıklığı (Time Complexity):

  - Rekürsif çözüm, her bir şehir kombinasyonunu incelemek için tüm şehirlerin kombinasyonunu tekrar tekrar hesaplar.
  - Her bir rekürsif çağrıda, şehir kombinasyonu bir şehir daha az olacak şekilde azaltılır.
  - Rekürsif çağrılar n kez yapıldığında ve her çağrı O(n) zaman alır.
  - Dolayısıyla, rekürsif çözümün zaman karmaşıklığı O(n!) olacaktır.

- Bellek Karmaşıklığı (Space Complexity):

  - Her bir rekürsif çağrı için bir çağrı yığını (call stack) kullanılır.
  - En derin rekürsif çağrı sayısı n'dir.
  - Dolayısıyla, rekürsif çözümün bellek karmaşıklığı O(n) olacaktır.

ÖZET

- Iteratif çözüm, zaman karmaşıklığı açısından rekürsif çözümden daha hızlıdır. Ancak, büyük n değerleri için hala pratik olmayabilir.
- Rekürsif çözüm, zaman karmaşıklığı açısından daha kötüdür, ancak bellek karmaşıklığı açısından daha iyi performans gösterir.
- Büyük veri kümeleri için daha verimli algoritmaların kullanılması gerekebilir.