Dijkstra Algoritması ve Nesne Tabanlı Programlama ile Geliştirilen Akıllı İzmit Ulaşım Sistemi

Ömer Faruk Özdemir
Onur Akbaş

***Özet*—Bu çalışmada, Kocaeli ili için geliştirilen akıllı ve kullanıcı odaklı bir toplu taşıma sistemi sunulmuştur. Sistem, farklı ulaşım türlerini (otobüs, tramvay, taksi) entegre eder ve kullanıcı tipi ile ödeme yöntemine göre en uygun rotaları hesaplar. Uygulamada, yönlü graflar üzerinde Dijkstra algoritması kullanılarak en kısa, en ucuz ve en hızlı rotalar oluşturulur. Sistem tamamen nesne tabanlı programlama prensipleriyle geliştirilmiş ve genişletilebilir bir yapıda tasarlanmıştır. Gerçek hayat verileriyle çalışan uygulama, rota hesaplamalarında yolcu indirimi, ödeme seçenekleri ve aktarma mantığı gibi şehir içi ulaşımı etkileyen tüm faktörleri kapsayacak şekilde geliştirilmiştir.**

I. Giriş

Günümüz şehir yaşamında, toplu taşıma sistemlerinin etkin kullanımı büyük önem taşımaktadır. Özellikle büyükşehirlerde, artan nüfus ve trafik sorunları nedeniyle kullanıcıların farklı ulaşım türleri arasında geçiş yapabilmesi kritik bir gerekliliktir. Bu durum, hem ekonomik hem de zamansal açıdan verimli çözümler gerektirmektedir.

Bu projede, Kocaeli ili özelinde, kullanıcı tipi ve ödeme şekline bağlı olarak değişen ücretlendirme politikalarıyla uyumlu, farklı ulaşım türlerini entegre eden bir rota hesaplama sistemi geliştirilmiştir. Sistem, .NET ve C# kullanılarak nesne yönelimli programlama (OOP) prensipleriyle inşa edilmiş olup, esnek ve genişletilebilir bir yapı sunmaktadır.

Geliştirilen sistem, kullanıcıya başlangıç ve hedef konumunu girdikten sonra en hızlı, en kısa veya en ekonomik güzergahı hesaplayarak sunmayı amaçlamaktadır. Bu amaçla, temel algoritma olarak Dijkstra yöntemi tercih edilmiştir.

II. Yöntem

A. Veri Modeli

Proje, veri girişi için `veri.json` dosyasını kullanmaktadır. Bu dosya, sistemin temel bileşenleri olan duraklar (konum bilgisi, tipi, bağlantıları), aktarma noktaları ve taksi ücret bilgilerini içermektedir. Her bir durak, yönlü bir grafa karşılık gelen bir düğüm (node) olarak modellenir. Duraklar arasındaki mesafe, süre ve ücret bilgileri ise kenar (edge) olarak temsil edilir. Böylece, sistemin veri yapısı gerçek hayattaki ulaşım ağını doğru biçimde yansıtır.

Veri modelinin esnekliği, sistemin farklı veri kaynaklarından gelen bilgileri de entegre edebilmesine olanak sağlar.

B. Nesne Tabanlı Tasarım

Sistem, SOLID prensiplerine uygun olarak nesne tabanlı bir mimari kullanılarak geliştirilmiştir. Bu yapı, bileşenlerin modüler ve bağımsız olarak güncellenebilmesini mümkün kılar. Temel sınıflar şunlardır:

- **Person (soyut sınıf):** Kullanıcı tipi; *General*, *Student* ve *ElderlyPerson* alt sınıfları ile temsil edilir.
- **Vehicle (soyut sınıf):** Ulaşım araçları; *Bus*, *Tram* ve *Taxi* sınıflarıyla modellenmiştir.
- **PaymentMethod:** Ödeme seçenekleri; *Cash*, *CreditCard*, *KentCard* gibi alt sınıflar aracılığıyla türetilmiştir.
- **RouteCalculator:** Kullanıcının başlangıç ve hedef konumuna göre en uygun güzergahı hesaplayan çekirdek sınıftır.

Bu yapı, sisteme yeni ulaşım türlerinin ve ödeme yöntemlerinin eklenmesini kolaylaştırır; mevcut bileşenlerde değişikliğe gerek kalmadan sistem genişletilebilir.

C. Kullanılan Algoritma: Dijkstra

Sistem, en uygun rotaları hesaplamak amacıyla Dijkstra algoritmasını kullanmaktadır. Bu algoritma, başlangıç düğümünden hedef düğüme olan en kısa yolu belirlemek için etkili bir yöntem sunar. Aşağıda algoritmanın yalancı kodu yer almaktadır:

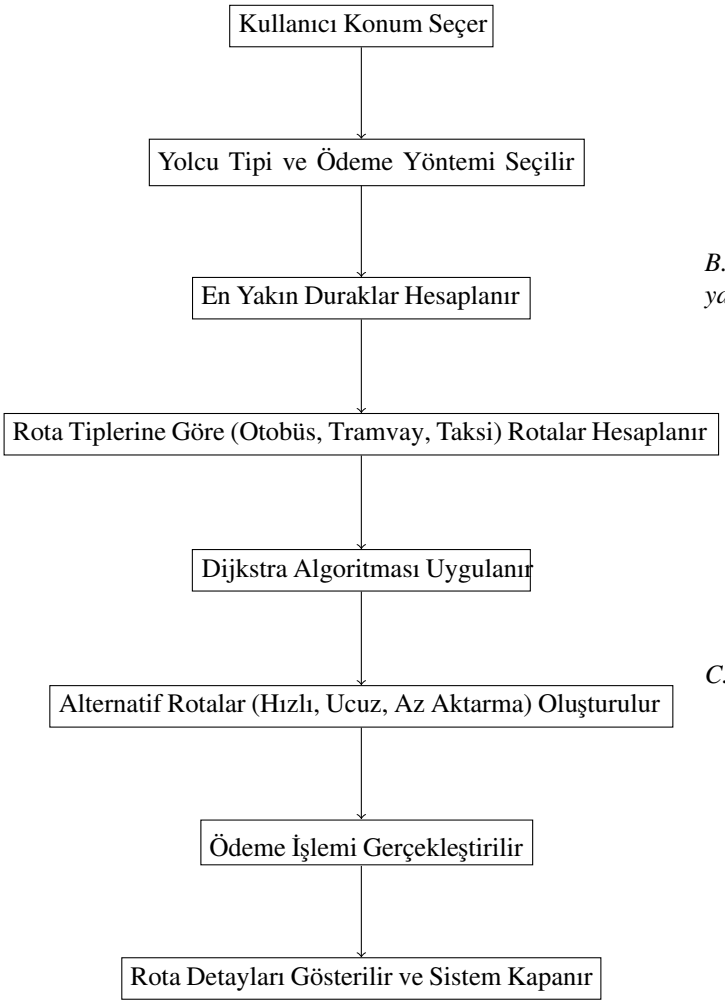
```
function Dijkstra(graph, startNode, endNode,
    costType):
    distance[] = sonsuz ile ba lat
    distance[startNode] = 0
    priorityQueue <- (0, startNode)
    previous[] <- null

    while priorityQueue is not empty:
        currentDistance, currentNode <- dequeue

        if currentNode == endNode:
            break

        foreach neighbor of currentNode:
            cost = getCost(neighbor, costType)
            newDistance = currentDistance + cost

            if newDistance < distance[neighbor]:
                distance[neighbor] = newDistance
                previous[neighbor] = currentNode
                enqueue(priorityQueue, (newDistance,
                      neighbor))

    return buildPath(previous, endNode)
```

Listing 1. Dijkstra Algoritmasının Yalancı Kodu

Bu algoritma, özellikle geniş veri setlerinde hızlı ve doğru sonuçlar üretmektedir.

D. Akış Diyagramı

Sistemin genel işleyişi, aşağıdaki akış diyagramında özetlenmiştir. Diyagram, her adımın net bir şekilde ayrıldığı ve okunabilirliği artıran bir düzenlemeyle sunulmaktadır:



Şekil 1. Sistemin genel işleyişini gösteren akış diyagramı.

III. Deneysel Sonuçlar

Geliştirilen sistem, çeşitli test senaryolarında dört farklı rota alternatifi sunmaktadır:

- **Otobüs ile Rota:** Öğrenci ve yaşlı kullanıcılar için en ekonomik seçenek.
- **Tramvay ile Rota:** Kısa sürede varış sağlayan alternatif.
- **Otobüs + Tramvay Kombinasyonu:** Transfer içeren ve esnek güzergah seçeneği.
- **Taksi ile Rota:** Doğrudan, hızlı ancak maliyetli seçenek.

Her rota, yolcu tipi indirim oranları, ödeme kontrolü, erişim yöntemi ile duraklar arası mesafe, süre ve ücret hesaplamaları dikkate alınarak belirlenmiştir. Test sonuçları, sistemin farklı senaryolarda tutarlı ve güvenilir performans sergilediğini göstermektedir.

IV. Geliştirici Notları ve Sunum Soruları

A. Soru 1: Tüm proje tamamlandıktan sonra yeni bir ulaşım aracı (örn: elektrikli scooter) eklendiğinde ne yapılmalı?

- `Vehicle` soyut sınıfı genişletilerek `Scooter` sınıfı oluşturulmalıdır.
- `CalculateFare()` ve `CalculateTravelTime()` metotları, scooter'a özel olarak yeniden tanımlanmalıdır.
- `veri.json` dosyasına scooter durakları eklenmeli (`"type": "scooter"`).
- `RouteCalculator` içerisinde, scooter duraklarını işleyip rota oluşturacak yeni bir metod eklenmelidir.
- Sistem, mevcut yapıyı değiştirmeden genişletilebilir.

B. Soru 2: Otonom taksiler projeye eklenmek istenirse ne yapılmalı?

- `AutonomousTaxi` adında yeni bir sınıf oluşturulmalı ve `Vehicle` sınıfından türetilmelidir.
- Kendi `OpeningFee` ve `CostPerKm` gibi parametreleri belirlenmelidir.
- Ücret yapısı ve hız sabiti, örneğin sabit rota süreleri uygulanarak farklılaştırılmalıdır.
- `veri.json` dosyasına bu araçlara özel veri yapısı (örn: `autonomousTaxiInfo`) eklenmelidir.
- `RouteCalculator` içerisinde, bu tür için rota hesaplama fonksiyonu yazılmalıdır.

C. Soru 3: Daha önce yazılmış hangi fonksiyonlar etkilenir?

- Yeni sınıf `Vehicle`'in mirası nedeniyle temel sınıflarda değişiklik gerekmez.
- `RouteCalculator` içerisinde yalnızca yeni ulaşım tipine özgü rota hesaplama fonksiyonları (örneğin, `CalculateScooterRoute()` ve `CalculateAutonomousTaxiRoute()`) eklenmelidir.
- `DisplayRoutes()` fonksiyonu, yeni ulaşım türünü tanıyacak şekilde güncellenmelidir.

D. Soru 4: Açık/Kapalı Prensibi nasıl uygulanır? Yeni ulaşım araçları mevcut fonksiyonlar değiştirilmeden nasıl eklenebilir?

- Sistem, `Vehicle` soyut sınıfı ve polymorphism kullanımıyla genişlemeye açıktır.
- Yeni araçlar, mevcut sistemi bozmadan eklenebilir.
- Kod, genişlemeye açık, değişime kapalı olacak şekilde tasarlanmıştır.
- Alternatif olarak, `IVehicle` gibi bir arayüz de kullanılabilir.

E. Soru 5: 65 yaş ve üzeri bireyler için ücretsiz seyahat hakkı 20 seyahat ile sınırlandırılırsa bu projeye nasıl eklenebilir?

- `ElderlyPerson` sınıfı içerisine bir sayaç (örn: `freeTripCount`) eklenmelidir.
- Bu sayaç, 20'ye kadar ücretsiz yolculuk hakkı tanımlar; sonrasında indirimli ücret uygulanır.

```
public class ElderlyPerson : Person {
    private int freeTripCount = 0;
    public override double GetDiscountFactor() {
        if (freeTripCount < 20) {
            freeTripCount++;
            return 0.0;
        }
        return 0.25;
    }
}
```

Listing 2. ElderlyPerson Sınıfı Örneği

Sadece `ElderlyPerson` sınıfı etkilenir. `Vehicle` sınıflarında indirim oranı zaten `GetDiscountFactor()` metodu ile alınmaktadır. `RouteCalculator` gibi diğer bileşenlerde değişikliğe gerek kalmamıştır.

V. Sonuç

Geliştirilen ulaşım sistemi, hem yazılım mimarisi hem de kullanıcı deneyimi açısından oldukça esnek ve geliştirilebilir bir yapıya sahiptir. Dijkstra algoritması ile yönlü graflar üzerinde rota hesaplaması yapılmakta, kullanıcıya alternatif güzergahlar sunulmakta ve tüm işlemler nesne tabanlı olarak yönetilmektedir. Yeni ulaşım türleri, farklı ücretlendirme modelleri ve belediye politikaları sisteme kolaylıkla entegre edilebilecek şekilde tasarlanmıştır.

Kocaeli ilinin ulaşım ağını gerçekçi biçimde temsil eden bu sistem, diğer şehirler için de örnek bir ulaşım platformu oluşturabilir. Gelecek çalışmalar kapsamında, sistemin performansının artırılması ve farklı veri setleriyle entegrasyonunun sağlanması hedeflenmektedir.

VI. Kaynakça

- E. Dijkstra, “A note on two problems in connexion with graphs,” *Numerische Mathematik*, 1959.
- Gamma et al., “Design Patterns: Elements of Reusable Object-Oriented Software,” 1994.
- Kocaeli Belediyesi Toplu Ulaşım Verileri, 2025.
- C# .NET Documentation, Microsoft Docs.