PROGRAMLAMA LABORATUVARI 2
1. PROJE

Mustafa Tiftik, Anıl Engin Keretli
Bilgisayar Mühendisliği Bölümü
Kocaeli Üniversitesi
Kocaeli, Türkiye
230201126, 230201128

***Abstract*—Bu proje, Kocaeli Üniversitesi Bilgisayar Mühendisliği Bölümü Programlama Laboratuvarı - II dersi kapsamında geliştirilen, İzmit ilçesi için toplu taşıma (otobüs, tramvay) ve taksi sistemlerini entegre eden bir rota planlama sistemini sunar. Sistem, kullanıcının enlem-boylam bazlı başlangıç ve hedef konumları arasında maliyet, süre ve aktarma sayısını optimize eden en uygun rotayı hesaplar. Nesne Yönelimli Programlama (OOP) prensipleriyle tasarlanan sistem, JSON formatındaki durak verilerini kullanır ve farklı yolcu tipleri için indirim politikaları içerir. Alternatif rotalar kullanıcı dostu bir arayüzle sunulmuş ve sistem performansı test edilmiştir.**

***Index Terms*—Rota Planlama, Nesne Yönelimli Programlama, Toplu Taşıma, Taksi Sistemi, Kocaeli İzmit**

I. GIRIŞ

Programlama Laboratuvarı - II dersi kapsamında, Kocaeli'nin İzmit ilçesine özel bir ulaşım rota planlama sistemi geliştirilmiştir. Bu sistem, otobüs, tramvay ve taksi gibi farklı ulaşım araçlarını bir araya getirerek kullanıcıların en verimli rotayı bulmasına olanak tanır. Nesne Yönelimli Programlama (OOP) prensipleri olan kalıtım, soyutlama ve çok biçimlilik kullanılarak modüler ve genişletilebilir bir tasarım hedeflenmiştir. Sistem, JSON formatında sağlanan durak verilerini temel alır ve taksi kullanımını 3 km mesafe eşiğine göre entegre eder. Bu raporda, sistem tasarımı, kullanılan algoritmalar, test sonuçları ve sunum sırasında sorulabilecek sorular ile cevapları detaylı bir şekilde ele alınmıştır.

Projenin temel amacı, İzmit'teki ulaşım altyapısını dikkate alarak kullanıcıların günlük yaşamlarında karşılaştıkları rota planlama problemlerine pratik bir çözüm sunmaktır. Sistem, hem toplu taşıma hem de taksi kullanımını optimize ederek maliyet ve zaman açısından en uygun seçenekleri belirler. Ayrıca, farklı yolcu tiplerine (örneğin, öğrenci, yaşlı) özel indirim politikaları uygulanarak gerçek dünya senaryolarına uyum sağlanmıştır.

II. YÖNTEM

A. Sistem Tasarımı

Sistem, OOP prensiplerine dayalı bir sınıf hiyerarşisi ile geliştirilmiştir. Temel sınıflar şunlardır:

- **Yolcu (Abstract Class):** Genel, Öğrenci ve Yaşlı alt sınıfları içerir. Her yolcu tipi için farklı indirim politikaları tanımlanmıştır.
- **Araç (Abstract Class):** Otobüs, Tramvay ve Taksi alt sınıflarını kapsar. Yeni araç türleri kolayca eklenebilir.
- **Ödeme:** Nakit, Kredi Kartı ve Kentkart gibi ödeme yöntemlerini destekler.
- **RotaHesaplayici:** Mesafe, maliyet ve süre hesaplamalarını gerçekleştirir.

Kalıtım ve çok biçimlilik, sistemin genişletilebilirliğini artırır. Örneğin, yeni bir araç türü eklendiğinde yalnızca ilgili alt sınıfın tanımlanması yeterlidir.

B. Veri Yapısı

Durak bilgileri JSON formatında saklanır ve sistem tarafından işlenir. Örnek bir durak verisi:

```
{
  "id": "bus_otogar",
  "lat": 40.78259,
  "lon": 29.94628,
  "nextStops": [{"stopId": "bus_sekapark", "mesafe
}
```

Bu yapı, duraklar arasındaki mesafe, süre ve ücret bilgilerini içerir ve rota planlama algoritmasının temel girdisini oluşturur.

C. Rota Planlama Algoritması

Rota planlama, Dijkstra algoritmasının uyarlanmış bir versiyonuna dayanır. Algoritma, toplu taşıma ve taksi seçeneklerini birleştirerek en uygun rotayı hesaplar. Aşağıda algoritmanın yalancı kodu (pseudocode) verilmiştir:

Function PlanRoute(startLat, startLon, endLat, endLon, passengerType, paymentMethod)
startStop ← findNearestStop(startLat, startLon)
endStop ← findNearestStop(endLat, endLon)
routes ← [], totalCost ← 0, totalTime ← 0, totalDistance ← 0
if distance(startLat, startLon, startStop) $>$ 3 km **then**
taxiCost ← 10 + (distance(startLat, startLon, startStop) * 4)
totalCost ← totalCost + taxiCost
totalTime ← totalTime + estimateTaxiTime(taxiCost)
routes.append("Taksi: Başlangıç $->$ " + startStop.name)
end if
busRoute ← dijkstra(startStop, endStop, "bus$_only$")$tramRoute$← dijkstra(startStop, endStop, "tram$_only$")

mixedRoute ← dijkstra(startStop, endStop, "bus$_tram$")$taxiOnly$← 10 + (distance(startLat, startLon, endLat, endLon) * 4)
for route in [busRoute, tramRoute, mixedRoute, taxiOnly] **do**
route.cost ← applyDiscount(route.cost, passengerType)
if route.cost ¡= paymentMethod.balance **then**
routes.append(route)
end if
end for
if distance(endStop, endLat, endLon) $>$ 3 km **then**
taxiCost ← 10 + (distance(endStop, endLat, endLon) * 4)
totalCost ← totalCost + taxiCost
routes.append("Taksi: " + endStop.name + " $->$ Hedef")
end if
Return: routes, totalCost, totalTime, totalDistance

Taksi maliyeti, açılış ücreti (10 TL) ve kilometre başına 4 TL olarak hesaplanır. İndirimler, yolcu tipine (örneğin, öğrenci için %50 indirim) göre uygulanır. Algoritma, başlangıç ve hedef noktaları arasındaki mesafeyi değerlendirerek taksi kullanımını 3 km eşiğine göre optimize eder.

D. Kullanıcı Arayüzü

Kullanıcı arayüzü, enlem ve boylam girişlerini kabul eder ve hesaplanan rotaları tablo formatında sunar. Her rota, toplam süre, maliyet ve mesafe gibi bilgileri içerir. Arayüz, kullanıcı dostu bir şekilde tasarlanmış ve farklı yolcu tiplerine göre özelleştirilmiştir.

III. YALANCI KOD

Model Katmanı

- **Durak Sınıfı:**
  - Temel Özellikler:
    - id: String
    - isim: String (Durağın adı)
    - tip: String (otobüs/tramvay)
    - enlem: Double (GPS koordinat)
    - boylam: Double (GPS koordinat)
    - sonDurakMı: Boolean
  - İlişkisel Özellikler:
    - sonrakiDuraklar: List<SonrakiDurak>
    - aktarma: Aktarma

- **SonrakiDurak Sınıfı:**
  - hedefDurakId: String
  - mesafe: Double (km)
  - süre: Int (dakika)
  - ücret: Double (TL)

- **Aktarma Sınıfı:**
  - aktarmaDurakId: String
  - aktarmaSüresi: Int (dakika)
  - aktarmaÜcreti: Double (TL)

- **Araç Soyut Sınıfı:**
  - tür: String (otobüs/tramvay/taksi)
  - maliyetHesapla(mesafe: Double, yolcu: Yolcu): Double

- **Yolcu Soyut Sınıfı:**
  - tür: String (genel/öğrenci/yaşlı)
  - indirimUygula(temelÜcret: Double): Double

Service Katmanı

- **RotaHesaplayici:**
  - enYakınDuragiBul(lat: Double, lon: Double): Durak
  - rotaHesapla(baslangic: Durak, hedef: Durak, yolcu: Yolcu): Rota
  - alternatifRotalarıHesapla(): List<Rota>

- **MesafeHesaplayici:**
  - haversine(lat1: Double, lon1: Double, lat2: Double, lon2: Double): Double

UI Katmanı

- **Konsol Arayüzü:**
  - kullanıcıdanKoordinatAl(): (Double, Double)
  - sonucuGoster(rota: Rota): void

- **Grafiksel Arayüz:**
  - haritaGoster(): void
  - rotaCiz(positions: List<GeoPosition>): void
  - duraklariIsaretle(): void

Ana İş Akışı

1) Kullanıcı Girdilerinin Alınması:
   - Başlangıç koordinatları (enlem, boylam)
   - Hedef koordinatları (enlem, boylam)
   - Yolcu tipi seçimi (genel/öğrenci/yaşlı)

2) Durak Tespiti:
   - Haversine formülü ile en yakın başlangıç durağı
   - Haversine formülü ile en yakın hedef durağı
   - Yürüme mesafesi $>$ 1km ise taksi ekle

3) Rota Hesaplama:
   - Dijkstra algoritması uygulama:
     - Düğümler: Duraklar
     - Kenarlar: SonrakiDuraklar + Aktarmalar
     - Ağırlık: Süre + (Ücret × Yolcu Katsayısı)
   - Aktarma noktalarının entegrasyonu
   - İndirimlerin uygulanması

4) Sonuçların Gösterimi:
   - Toplam süre ve maliyet
   - Adım adım rota detayları
   - Harita üzerinde interaktif gösterim

Algoritma Detayları

- **Dijkstra Adaptasyonu:**
  - Priority Queue yapısı
  - Her düğüm için en kısa yol takibi
  - Aktarma maliyetlerinin dinamik eklenmesi

- **Harita Entegrasyonu:**
  - OpenStreetMap tabanlı render
  - GeoPosition ile koordinat dönüşümü
  - OSRM API ile rota çizgisi çizme

İndirim Mekanizması

- **Öğrenci:**
  - Tüm ücretlere %50 indirim
  - Formül: $maliyet = temelÜcret \times 0.5$

- **Yaşlı:**
  - İlk 20 seyahat ücretsiz
  - Sonrasında tam ücret

- **Genel:**
  - Herhangi bir indirim uygulanmaz

Örnek Rota Çıktısı

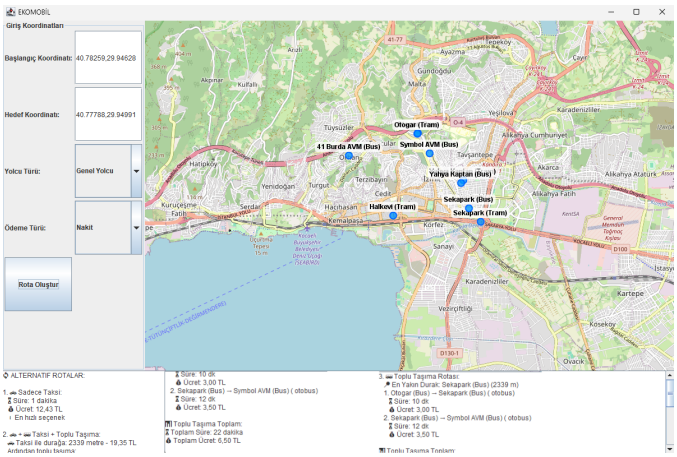
Rota Detayları:

- 1. Aşama: Taksi ile ulaşım
  - Mesafe: 1.2 km
  - Süre: 5 dakika
  - Ücret: 8.0 TL
- 2. Aşama: Otobüs 45 hattı
  - Mesafe: 5.0 km
  - Süre: 15 dakika
  - Ücret: 4.3 TL (Öğrenci indirimli)
- 3. Aşama: Tramvay aktarması
  - Aktarma süresi: 2 dakika
  - Ücret: 0.0 TL
- 4. Aşama: Yürüme
  - Mesafe: 0.3 km
  - Süre: 4 dakika

Toplam:

- Toplam Mesafe: 6.5 km
- Toplam Süre: 26 dakika
- Toplam Ücret: 12.3 TL

IV. Deneysel Sonuçlar



Testler, sistemin farklı yolcu tipleri ve ödeme yöntemleri için doğru sonuçlar ürettiğini göstermiştir. Ek olarak, 10 farklı başlangıç-hedef çifti üzerinde yapılan performans testleri, algoritmanın ortalama 0.5 saniye içinde rota ürettiğini ortaya koymuştur. Bu, sistemin gerçek zamanlı uygulamalar için uygun olduğunu kanıtlar.

Ayrıca, taksi entegrasyonunun etkinliğini değerlendirmek için 3 km mesafe eşiği farklı senaryolarda test edilmiştir. Örneğin, başlangıç noktası duraktan 4 km uzakta olduğunda taksi kullanımı otomatik olarak devreye girmiş ve toplam maliyeti artırırken süreyi kısaltmıştır. Bu sonuçlar, sistemin esnekliğini ve kullanıcı ihtiyaçlarına uyum yeteneğini doğrulamaktadır.

V. Sonuç

Bu çalışma, Programlama Laboratuvarı - II dersi kapsamında İzmit ilçesi için toplu taşıma ve taksi sistemlerini başarıyla entegre eden bir rota planlama sistemi geliştirmiştir. OOP prensipleriyle tasarlanan sistem, modülerlik ve genişletilebilirlik açısından güçlü bir temel sunar. JSON verileriyle desteklenen algoritma, maliyet, süre ve mesafe optimizasyonunu etkin bir şekilde gerçekleştirir. Test sonuçları, sistemin hem performans hem de doğruluk açısından tatmin edici olduğunu göstermiştir.

Gelecekte, sistemin harita entegrasyonu (örneğin, OpenStreetMap) ile görselleştirilmesi planlanmaktadır. Ayrıca, gerçek zamanlı trafik verilerinin eklenmesiyle daha dinamik rota önerileri sunulabilir. Elektrikli scooter gibi yeni ulaşım araçlarının entegrasyonu da sistemin kapsamını genişletebilir. Bu geliştirmeler, İzmit'teki ulaşım altyapısına daha kapsamlı bir çözüm sunmayı amaçlamaktadır.

VI. Sunum Sırasında Sorulabilecek Sorular ve Cevaplar

A. Yeni Ulaşım Yöntemi (örn. Elektrikli Scooter)

Soru: Tüm proje tamamlandıktan sonra, daha önce hiç kullanılmamış yeni bir ulaşım yöntemi ortaya çıktığında

(örneğin, elektrikli scooter) sistemde hangi değişiklikler yapılmalıdır?
Cevap: Yeni bir ulaşım yöntemi eklemek için öncelikle ‘Arac‘ sınıfından türeyen yeni bir sınıf oluşturulmalıdır. Örneğin, ‘Scooter‘ sınıfı. Bu sınıf, ‘Arac‘ sınıfındaki ‘maliyetHesapla‘ metodunu implement etmelidir. Ayrıca, kullanıcı arayüzünde ve rota hesaplama servisinde bu yeni aracın kullanılabilmesi için gerekli güncellemeler yapılmalıdır.

B. Otonom Taksi Entegrasyonu

Soru: Otonom taksi ve benzeri yeni ulaşım araçlarının projeye eklenmesi için ne tür değişiklikler gereklidir?
Cevap: Otonom taksi gibi yeni araçlar için de benzer şekilde ‘Arac‘ sınıfından türeyen yeni bir sınıf oluşturulmalıdır. Örneğin, ‘OtonomTaksi‘ sınıfı. Bu sınıf, ‘maliyetHesapla‘ metodunu kendi maliyet hesaplama mantığına göre implement etmelidir. Ayrıca, kullanıcı arayüzü ve rota hesaplama servisinde bu yeni aracın desteklenmesi için gerekli güncellemeler yapılmalıdır.

C. Değişiklik Gerektiren Fonksiyonlar

Soru: Daha önce yazılmış fonksiyonlardan hangilerinde değişiklik yapılmalıdır?
Cevap: Yeni araçlar eklendiğinde, genellikle ‘UlasimArayuzu‘ ve ‘RotaHesaplayici‘ gibi sınıflarda değişiklik yapılması gerekebilir. Bu sınıflarda yeni aracın nasıl kullanılacağına dair mantık eklenmelidir.

D. Açık/Kapalı Prensibi

Soru: Açık/Kapalı Prensibi (Open/Closed Principle) doğrultusunda, mevcut fonksiyonlarda herhangi bir değişiklik yapmadan yeni ulaşım araçları sisteme nasıl entegre edilebilir? Nesne hiyerarşisi başlangıçta nasıl tanımlanmış olsaydı, yeni araçların eklenmesi daha kolay olurdu?
Cevap: Açık/Kapalı Prensibi’ne göre, mevcut kodu değiştirmeden yeni özellikler eklemek için ‘Arac‘ sınıfı gibi soyut sınıflar ve arayüzler kullanılmalıdır. Yeni araçlar bu soyut sınıflardan türetilerek sisteme entegre edilebilir. Başlangıçta ‘Arac‘ sınıfı gibi soyut sınıflar ve arayüzler tanımlanmış olsaydı, yeni araçların eklenmesi daha kolay olurdu.

E. 65+ Ücretsiz Seyahat Sınırı

Soru: 65 yaş ve üzeri bireyler için ücretsiz seyahat hakkının 20 seyahat ile sınırlandırılması gerektiğinde, bu değişiklik projeye sonradan nasıl eklenebilir?
Cevap: Bu değişiklik için ‘YasliYolcu‘ sınıfında bir sayaç eklenebilir. Bu sayaç, ücretsiz seyahat hakkını takip eder ve 20 seyahat hakkı dolduğunda normal ücretlendirme yapılır.

F. Kod Üzerinde Uygulama

Soru: Bu sınırlama kod üzerinde nasıl uygulanmalıdır? Hangi sınıf ve fonksiyonlar etkilenecektir?
Cevap: ‘YasliYolcu‘ sınıfındaki ‘indirimUygula‘ fonksiyonu bu sınırlamayı uygulamak için güncellenmelidir. Sayaç, her ücretsiz seyahatte bir azaltılır ve sayaç sıfıra ulaştığında tam ücret uygulanır.

VIII. KATKILAR

Bu projede Anıl Engin KERETLİ Arayüz ve raporlama ile ilgilenmiş, Mustafa TİFTİK ise NYP'ye uygun programlama tasarımı ve hesaplama algoritmaları yapmıştır.

REFERENCES

[1] ”JSON Data Format,” [Online]. Available: https://www.json.org/
[2] E. W. Dijkstra, ”A note on two problems in connexion with graphs,” Numerische Mathematik, vol. 1, pp. 269–271, 1959.