Prolab2 1. Proje
İzmit Ulaşım Uygulaması

Yunus Emre Arı
230201096
Bilgisayar Mühendisliği
Kocaeli Üniversitesi
Kocaeli, Türkiye

Elifnur Caştur
210201002
Bilgisayar Mühendisliği
Kocaeli Üniversitesi
Kocaeli, Türkiye

***Özetçe*—Bu çalışma, İzmit şehri için geliştirilmiş bir ulaşım rota planlama sistemini sunmaktadır. Sistem, kullanıcıların belirtilen başlangıç ve hedef noktaları arasında otobüs, tramvay, taksi ve yürüme seçeneklerini kullanarak optimum rotaları hesaplamaktadır. Çalışmada, rota hesaplama algoritmaları, aktarma noktalarının yönetimi, özel günlerde ücretsiz toplu taşıma simülasyonu ve kullanıcı arayüzü tasarımı ele alınmıştır. Sistem, farklı rota kombinasyonlarını değerlendirerek kullanıcılara mesafe, süre ve maliyet açısından en uygun alternatifleri sunmaktadır.**

I. Giriş

Modern şehirlerde toplu taşıma sistemlerinin karmaşıklığı, kullanıcıların en verimli rotaları belirlemesini zorlaştırmaktadır. Farklı ulaşım seçenekleri, duraklar arası mesafeler, aktarma noktaları ve değişken ücretlendirme yapıları, yolculuk planlamasında önemli faktörlerdir. İzmit Ulaşım Rota Planlama Sistemi, bu sorunlara çözüm getirmek amacıyla geliştirilmiş, kullanıcı dostu bir yazılımdır.

Bu sistem, İzmit şehrindeki otobüs ve tramvay hatlarını, durakları, aktarma noktalarını ve taksi hizmetlerini birleştirerek çok modlu bir ulaşım ağı oluşturmaktadır. Kullanıcılar, başlangıç ve hedef noktalarını koordinat olarak girebildikleri gibi, doğrudan durak seçerek de rota hesaplaması yapabilmektedir. Ayrıca sistem, yolcu tipine göre (normal, öğrenci, yaşlı) farklı ücretlendirme modelleri sunmakta ve resmi bayram günlerinde ücretsiz toplu taşıma hizmetlerini simüle edebilmektedir.

SİSTEM MİMARİSİ

Genel Yapı

İzmit Ulaşım Rota Planlama Sistemi, Java programlama dili kullanılarak geliştirilmiş ve nesne yönelimli tasarım prensipleri uygulanmıştır. Sistem, aşağıdaki ana bileşenlerden oluşmaktadır:

1. **Veri Yönetimi**: JSON formatında saklanan şehir verileri, duraklar, hatlar ve ücretlendirme bilgileri

2. **Graf Yapısı**: Duraklar ve bağlantıları temsil eden veri yapısı

3. **Rota Hesaplama Motoru**: Dijkstra algoritması tabanlı en kısa yol hesaplaması

4. **Aktarma Noktası Yöneticisi**: Farklı ulaşım modları arasındaki aktarma noktalarını yöneten modül

5. **Kullanıcı Arayüzü**: Swing kütüphanesi ile geliştirilmiş grafiksel arayüz

6. **Tarih Simülasyonu**: Bayram günleri ve özel günlerde ücretsiz ulaşım simülasyonu

ALGORİTMA VE METODOLOJİ

Veri Modeli

Sistemde kullanılan veri modeli, durakları düğüm (node) ve duraklar arası bağlantıları kenar (edge) olarak temsil eden bir graf yapısıdır. Her durak, kimlik bilgisi, isim, tür (otobüs veya tramvay), coğrafi koordinatlar ve bağlantılı olduğu diğer duraklar gibi bilgileri içerir. Duraklar arası bağlantılar ise mesafe, süre ve ücret bilgilerini içerir.

Aktarma noktaları, farklı ulaşım modları arasında geçiş yapılabilen özel düğümler olarak modellenmiştir. Bu noktalar, aktarma süresi ve aktarma ücreti gibi ek bilgileri içerir.

Algoritma

Projede kullanılan yol bulma algoritması, **Dijkstra** Algoritmasıdır. Bu algoritma, bir grafikteki düğümler (nodes) arasında en kısa yolu bulmak için kullanılır. Algoritma, başlangıç düğümünden diğer tüm düğümlere olan en kısa mesafeyi hesaplar ve hedef düğüme ulaşıldığında işlemi sonlandırır.

Algoritmanın Çalışma Mantığı

Giriş**:**

- Bir grafik (düğümler ve kenarlar).
- Başlangıç düğümü (start node).
- Hedef düğüm (end node).

Çıkış:

- Başlangıç düğümünden hedef düğüme kadar olan en kısa yol.
- Bu yolun toplam mesafesi.

Adımlar**:**

- Tüm düğümler için başlangıçta mesafeler sonsuz (Infinity) olarak ayarlanır.
- Başlangıç düğümünün mesafesi 0 olarak ayarlanır.
- Ziyaret edilmemiş düğümler bir öncelik kuyruğuna eklenir.
- Kuyruktan en düşük mesafeye sahip düğüm seçilir.
- Bu düğümün komşuları ziyaret edilir ve mesafeler güncellenir:
  - Eğer yeni mesafe, mevcut mesafeden daha kısa ise, mesafe güncellenir.
- Hedef düğüme ulaşıldığında işlem sonlandırılır.

---

1. Yeni Bir Ulaşım Yöntemi (Örneğin, Elektrikli Scooter) Eklendiğinde

a. Sistem Üzerinde Yapılması Gereken Değişiklikler
- Yeni bir Vehicle türü eklenmelidir.
  - Elektrikli scooter gibi yeni bir ulaşım yöntemi için Vehicle sınıfını genişleten bir alt sınıf oluşturulmalıdır.
- Yeni ulaşım yöntemi için rota hesaplama mantığı eklenmelidir.
  - Örneğin, scooter için mesafe, süre ve ücret hesaplama mantığı tanımlanmalıdır.

b. Etkilenecek Sınıflar ve Fonksiyonlar
- Vehicle Sınıfı:
  - Yeni bir alt sınıf (örneğin, Scooter) oluşturulmalıdır.
- GraphBuilder Sınıfı:
  - Yeni ulaşım yöntemi için rota hesaplama fonksiyonları eklenmelidir (örneğin, calculateScooterOnlyRoute).
- Route Sınıfı:
  - Yeni ulaşım türü için RouteType sabitleri eklenmelidir (örneğin, ONLY_SCOOTER).

c. Açık/Kapalı Prensibi (Open/Closed Principle)
- Eğer Vehicle sınıfı ve rota hesaplama mantığı başlangıçta polimorfizm ile tasarlanmış olsaydı, yeni bir ulaşım yöntemi eklemek için mevcut kodda değişiklik yapmaya gerek kalmazdı.
- Örneğin:
  - Vehicle sınıfı bir abstract class veya interface olarak tanımlanabilir.
  - Her araç türü (Bus, Tram, Taxi, Scooter) bu sınıfı genişletir ve kendi hesaplama mantığını uygular.

2. Otonom Taksi ve Benzeri Araçların Eklenmesi

a. Gerekli Değişiklikler
- Yeni bir Vehicle türü eklenmelidir.
  - Otonom taksi için AutonomousTaxi sınıfı oluşturulabilir.
- Ücret ve süre hesaplama mantığı eklenmelidir.
  - Otonom taksi için özel ücretlendirme ve süre hesaplama mantığı tanımlanmalıdır.

b. Etkilenecek Sınıflar
- Vehicle Sınıfı:
  - Yeni bir alt sınıf (AutonomousTaxi) eklenir.
- GraphBuilder Sınıfı:
  - Otonom taksi için rota hesaplama fonksiyonu eklenir (örneğin, calculateAutonomousTaxiRoute).

c. Açık/Kapalı Prensibi
- Eğer GraphBuilder sınıfı rota hesaplama işlemlerini polimorfizm ile yönetiyor olsaydı, yeni bir araç eklemek için mevcut kodda değişiklik yapmaya gerek kalmazdı.
- Örneğin:
  - GraphBuilder sınıfı, rota hesaplama işlemini Vehicle sınıfına devredebilir

3. 65 Yaş ve Üzeri Bireyler İçin Ücretsiz Seyahat Sınırlandırması

a. Gerekli Değişiklikler
- Seyahat sayısını takip etmek için bir sayaç eklenmelidir.
  - Örneğin, Passenger sınıfına bir travelCount alanı eklenebilir.
- Seyahat sınırını kontrol eden bir mekanizma eklenmelidir.
  - Örneğin, Passenger sınıfında bir canTravelForFree metodu tanımlanabilir.

c. Etkilenecek Sınıflar ve Fonksiyonlar

- Passenger Sınıfı:
  - Seyahat sayısını takip etmek için bir sayaç eklenir.
  - Ücretsiz seyahat hakkını kontrol eden bir mekanizma eklenir.
- Rota Hesaplama Fonksiyonları:
  - Ücret hesaplama sırasında Passenger.calculateDiscount metodu çağrılır.

---

4. Yeni Araçların Daha Kolay Eklenmesi için Nesne Hiyerarşisi

Eğer başlangıçta aşağıdaki gibi bir nesne hiyerarşisi tanımlanmış olsaydı, yeni araçların eklenmesi çok daha kolay olurdu:

- Vehicle Sınıfı:
  - Tüm araç türleri için ortak davranışları tanımlar.
- Alt Sınıflar:
  - Bus, Tram, Taxi, Scooter, AutonomousTaxi gibi her araç türü için ayrı sınıflar.

Projenin Genel Akışı

1. Kullanıcı Girişi:
   - Kullanıcı başlangıç ve hedef noktalarını seçer (koordinat veya durak).
   - Kullanıcı yolcu türünü (Genel, Öğrenci, Yaşlı) ve ödeme yöntemini (Nakit, Kredi Kartı, Kentkart) seçer.
2. Rota Hesaplama:
   - Başlangıç ve hedef noktalarına en yakın duraklar bulunur.
   - Duraklar arasındaki en kısa yol hesaplanır (Dijkstra algoritması).
   - Rota türü belirlenir (Sadece Otobüs, Yürüme + Otobüs, vb.).
3. Ücret Hesaplama:
   - Yolcu türüne göre indirim uygulanır.
   - Ödeme yöntemine göre ek ücretler veya indirimler hesaplanır.
4. Sonuç Gösterimi:
   - Kullanıcıya rota özeti (mesafe, süre, ücret) ve detayları gösterilir.
   - Alternatif rotalar listelenir.

Akış Diyagramı Taslağı
Başlangıç

1. Kullanıcı başlangıç ve hedef noktalarını girer.
   - [Karar]: Koordinat mı yoksa durak mı?
     - Koordinat: Kullanıcı koordinatları girer.
     - Durak: Kullanıcı durak listesinden seçim yapar.
2. Kullanıcı yolcu türünü seçer.
   - Genel, Öğrenci, Yaşlı.
3. Kullanıcı ödeme yöntemini seçer.
   - Nakit, Kredi Kartı, Kentkart.

Rota Hesaplama

4. Başlangıç ve hedef noktalarına en yakın duraklar bulunur.
   - findNearestStop veya findNearestBusStop.
5. Duraklar arasındaki en kısa yol hesaplanır.
   - findShortestPath (Dijkstra algoritması).
6. Rota türü belirlenir.
   - Sadece Otobüs, Yürüme + Otobüs, vb.

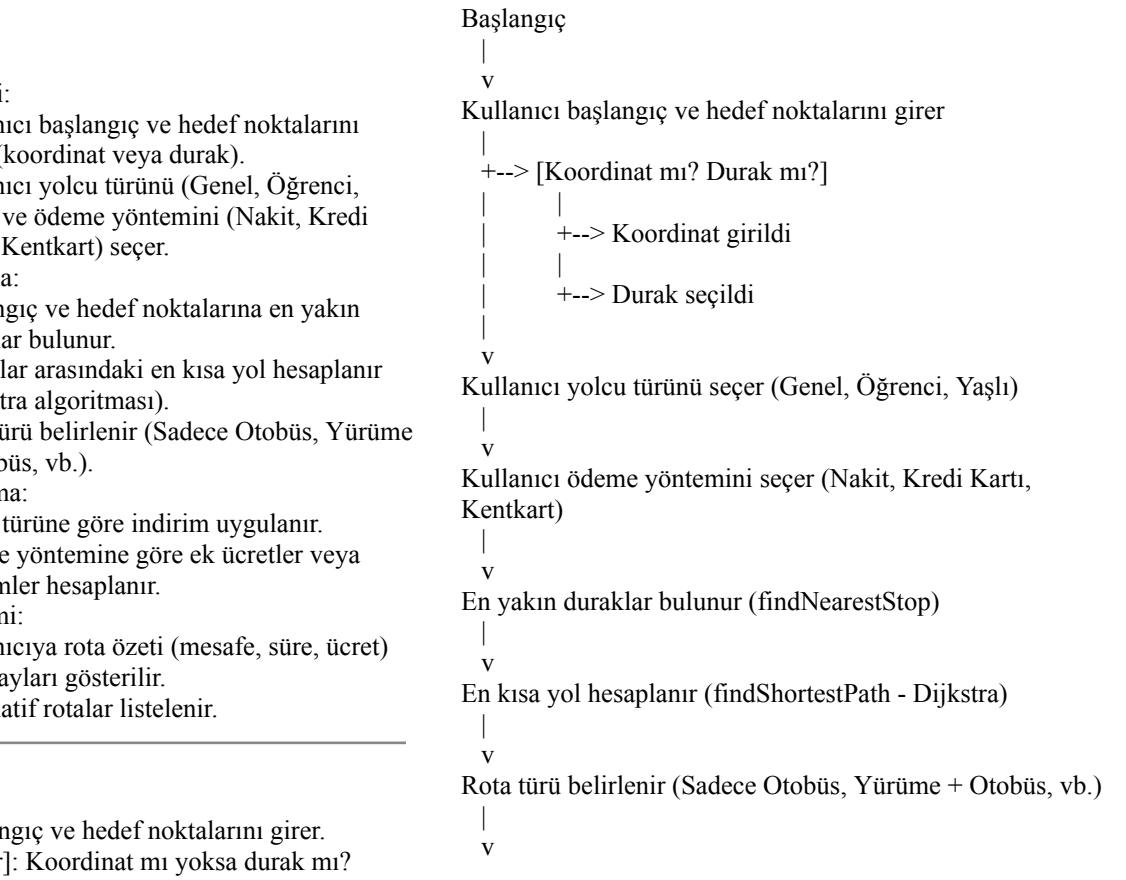
Ücret Hesaplama

7. Yolcu türüne göre indirim hesaplanır.
   - Passenger.calculateDiscount.
8. Ödeme yöntemine göre ek ücretler uygulanır.
   - Örneğin, kredi kartı ile ödeme için ek ücret.

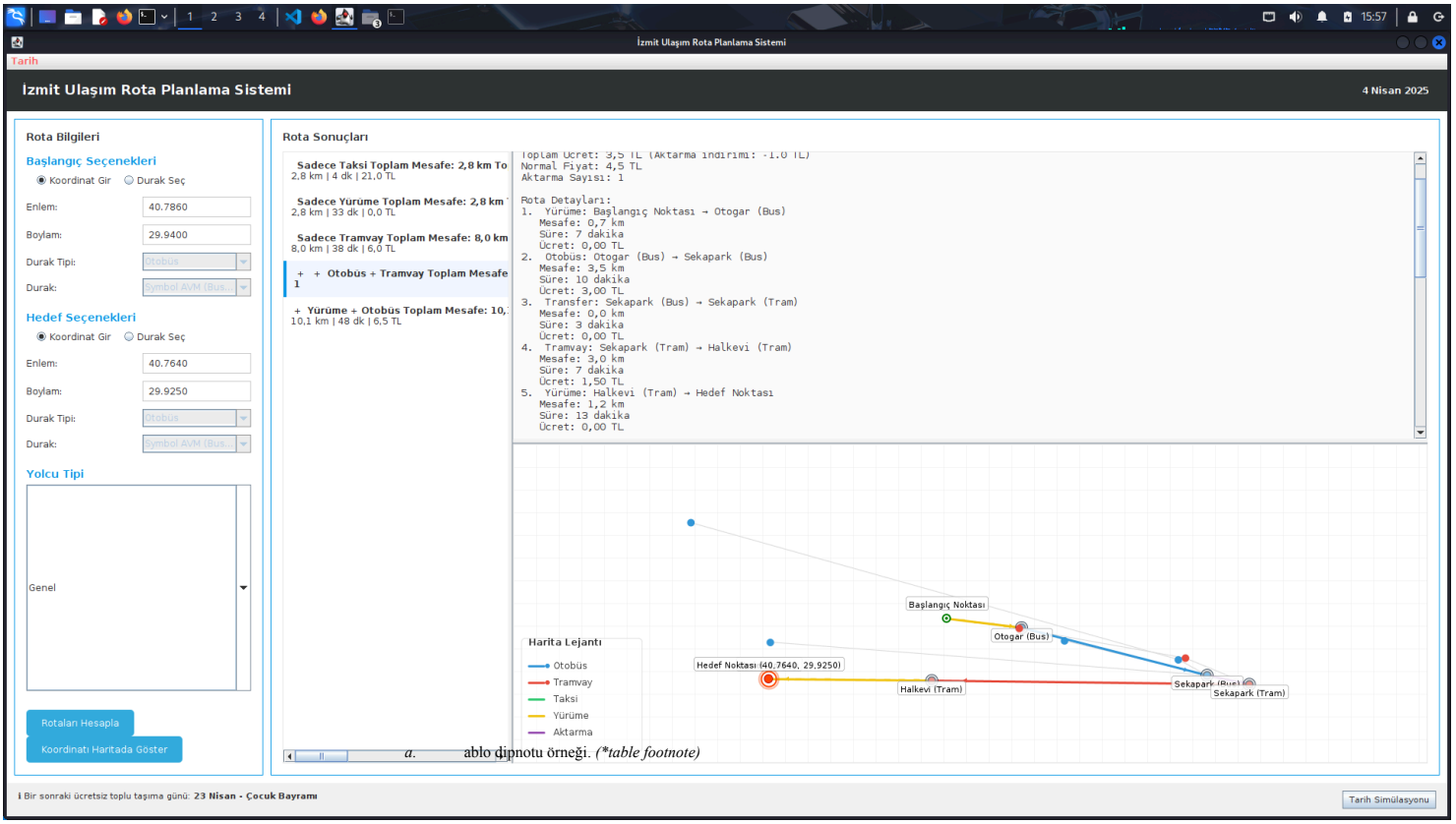
Sonuç Gösterimi

9. Kullanıcıya rota özeti ve detayları gösterilir.
   - Mesafe, süre, ücret.
10. Alternatif rotalar listelenir.

Bitiş



Yolcu türüne göre indirim hesaplanır
(Passenger.calculateDiscount)
  |
  v
Ödeme yöntemine göre ek ücretler uygulanır
  |
  v
Rota özeti ve detayları gösterilir
  |
  v
Alternatif rotalar listelenir
  |
  v
Bitiş



a. ablo dipnotu örneği. *(*table footnote)*

KAYNAKLAR.

[1] [4] Oracle. (2023). "Java Swing Documentation". [Online]. Available: https://docs.oracle.com/javase/tutorial/uiswing/

[2] Google. (2023). "Gson Documentation". [Online]. Available: https://github.com/google/gson

[3] Dijkstra, E. W. (1959). "A note on two problems in connexion with graphs". Numerische Mathematik, 1(1), 269-271.