Soru 1. Üniversitenizin kampüsünde sürekli kaybolan yeni öğrenciler için, "kampüs navigasyon sistemi" geliştirdiğiniz bir projede çalıştığınızı düşünün. Sistem, kampüsteki binalar ve bu binaları birbirine bağlayan yollar üzerinden, bir binadan diğerine en az "geçiş noktası" ile nasıl gidileceğini bulmalı. Ayrıca kampüste yürüyerek gidilemeyen kopuk alanlar olup olmadığını da analiz etmek istiyorsunuz.

Görevler:

1. **Graf Modeli Kurulumu**
   - Kampüsteki en az **8–10 farklı bina/konum** belirleyin (ör: Mühendislik ve Doğa Bilimleri Fakültesi, Kütüphane, Yemekhane, Yurt, Spor Salonu vb.).
   - Bu binaları **düğüm (node)**, aralarındaki yürünebilir yolları **kenar (edge)** olarak modelleyin.
   - Grafı hem **komşuluk matrisi** hem de **komşuluk listesi** ile temsil eden veri yapıları tasarlayın.
2. **BFS ile En Az Duraklı Yol Bulma**
   - Kullanıcıdan bir **başlangıç binası** ve bir **hedef bina** alın.
   - BFS algoritmasını kullanarak, kenar sayısı cinsinden **en az duraklı** yolu bulun.
   - Bulduğunuz yolu, düğüm sırası ile ekranda gösterin (**örn:** Mühendislik → Merkez Kantin → Kütüphane).
3. **DFS ile Bileşen Kontrolü**
   - DFS kullanarak grafın **bağlı bileşenlerini** bulun.
   - Kampüste yürüyerek ulaşılamayan kopuk alanlar (varsa) olup olmadığını tespit edin.
   - Bu bileşenleri sözel olarak yorumlayın.
4. **Görsel/Diyagram Hazırlama**
   - Kampüsü temsil eden basit bir **graf diyagramı** hazırlayın (elle çizim + fotoğraf, dijital çizim veya sunum içi şema olabilir).
   - Diyagram üzerinde düğümleri ve kenarları açıkça gösterin.
5. **Analiz**
   - BFS ve DFS algoritmalarının **zaman karmaşıklığını (Big-O)** kısaca açıklayın.

Soru 2. Modern web tarayıcılarında, kullanıcılar ziyaret ettikleri sayfalar arasında **geri (Back)** ve **ileri (Forward)** butonlarıyla gezinebilir. Siz de basit bir tarayıcı geçmişi simülatörü geliştiriyorsunuz. Kullanıcı yeni bir siteye gittiğinde, geri/ileri yığınlarının (veya kuyruklarının) nasıl değiştiğini veri yapıları ile göstermeniz gerekiyor.

Görevler:

1. **Veri Yapılarının Tasarımı**
   - **Aktif sayfa**, **Back yığını** ve **Forward yığını** (veya kuyruğu) içeren bir yapı tasarlayın.
   - Her sayfa en azından bir **URL (string)** bilgisine sahip olmalıdır.
2. **Temel İşlemler**
   - visit(url): Yeni bir siteye gitme işlemi
   - Aktif sayfayı Back yığınına ekleyip yeni URL'yi aktif yapın.
   - Forward yığını temizlenmelidir.
   - back(): Geri gitme işlemi
   - Mümkünse aktif sayfayı Forward yığınına atın, Back yığından bir önceki sayfayı çekip aktif yapın.
   - forward(): İleri gitme işlemi
   - Mümkünse aktif sayfayı Back yığınına atın, Forward yığından bir sonraki sayfayı çekip aktif yapın.
3. **Kullanıcıdan Girdi Alma**
   - Kullanıcıya basit bir menü sunun:
     a. Yeni URL ziyaret et
     b. Back
     c. Forward
     d. Çıkış
   - Kullanıcının yaptığı işlemler sonucunda Back ve Forward yığınlarının o anki durumunu ekrana yazdırın.
4. **Yığın İçeriklerinin Adım Adım Gösterimi**
   - Sunumda, örnek bir senaryo seçin:
   - **Örn:** google.com → youtube.com → github.com → back → back → forward gibi.
   - Bu senaryonun her adımında:
   - Aktif sayfa
   - Back yığını içeriği
   - Forward yığını içeriğini tablo veya şekil ile gösterin.
5. **Analiz**
   - Yığın kullanımının neden bu problem için uygun olduğunu açıklayın.
   - Zaman karmaşıklığını (push/pop işlemleri için) kısaca tartışın.

Teknik Beklentiler:

- Soruları temiz bir A4 kağıdına **kendi el yazınız** ile yazınız. Teslim düzenine dikkat edeniz.
- Uygulama **istediğiniz kodlama dili** ile yazılmalıdır (Teknik sıkıntı yaşanmaması adına, kişisel bilgisayarlarınız yanınızda ve çalışır halde bulunsun. Kodlarınızın görülmesi ve/veya kod yazmanız gerekebilir.).
- **1. Soru için:** Küçük test senaryoları (en az 3 farklı başlangıç/hedef çifti) deneyip sonuçları raporlayın.
- **2. Soru için:** En az bir örnek senaryo, ekran çıktısı şeklinde sunulmalıdır.
- Kodda fonksiyonel bir yapı olmalıdır (işlemler fonksiyonlara ayrılmalıdır.).