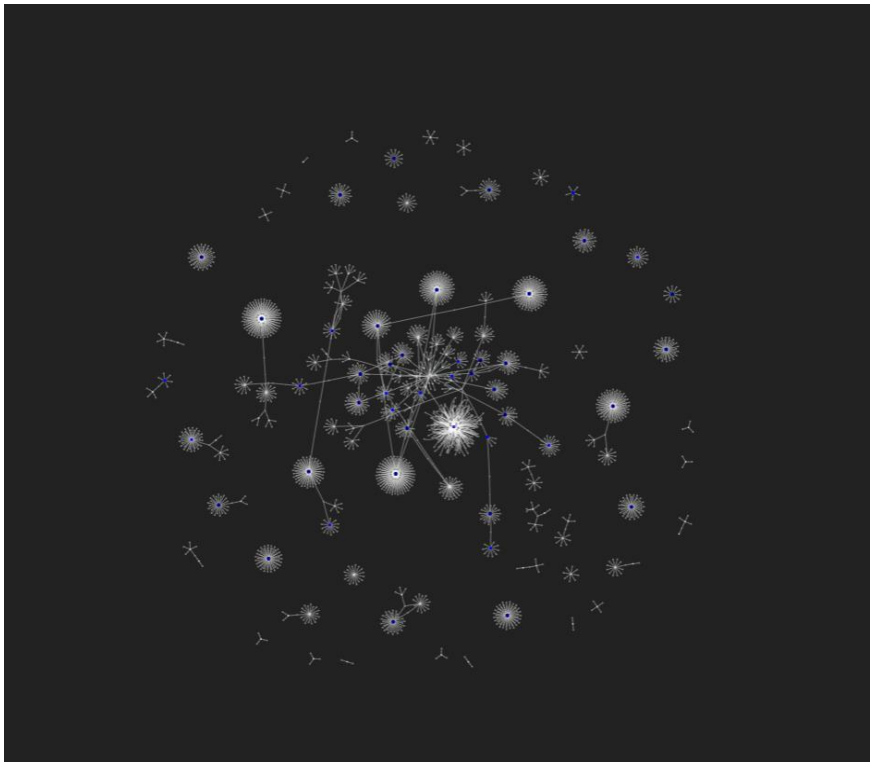
Programlama Labaratuvarı 3.Proje Raporu

Sadık Gölpek
sadikgolpek@gmail.com

Abdullah Önder
abdullahonder726@gmail.com



a) Graf yapımız.

I.Özet

Bu proje, akademik bir veri seti kullanılarak yazarlar arasındaki iş birliği ilişkilerini modelleyen bir graf yapısı oluşturmayı ve bu yapı üzerinde çeşitli veri yapısı ve algoritma konseptlerini uygulamayı amaçlamaktadır. Proje kapsamında, öğrencilerden hem teorik bilgilerini hem de uygulamalı becerilerini geliştirmeleri beklenmiştir.

Proje boyunca şu hedefler gözetilmiştir:

Graf Veri Yapısını Öğrenmek: Yazarların düğümleri, iş birliği ilişkilerinin ise kenarları temsil ettiği bir graf yapısı oluşturulmuştur.
Veri Yapısı Kavramlarını Uygulamak: Graf üzerinde arama ve sıralama gibi işlemler gerçekleştirilmiştir.
Gerçek Dünyadan Problem Çözmek: Akademik iş birliği gibi somut bir problem üzerinden veri yapılarını uygulamalı bir şekilde öğrenme imkanı sağlanmıştır.
Grafiksel Temsiller ile Çalışmak: Grafın görselleştirilmesi ve analiz sonuçlarının etkili bir şekilde sunulması amaçlanmıştır.

Proje için belirli bir programlama dili zorunluluğu bulunmamakla birlikte, biz görselleştirmede kütüphane desteğinden dolayı Python kullanmayı tercih ettik. Veriler Excel dosyalarından çekilmiş, bu veriler uygun şekilde işlenerek bir graf modeli oluşturulmuştur. Bu süreçte aşağıdaki araç ve kütüphaneler kullanılmıştır:

- **Pandas:** Excel dosyalarından veri çekmek ve işlemek için.
- **os:** Sistem dosyalarını kontrol etmek ve yönetmek için.
- **Pyvis:** Graf yapısının görselleştirilmesi için.
- **webview API:** Dinamik veri akışını sağlamak için.
- **HTML, CSS, ve JavaScript:** Verilerin web tabanlı görselleştirilmesi ve dinamik sunumları için.

Proje kapsamında, verilen 7 farklı işlevsel isteği yerine getirmek için manuel algoritmalar geliştirilmiş ve hazır kütüphaneler kullanılmamıştır. Bu, öğrencilerin algoritma geliştirme becerilerini pekiştirmeyi hedeflemiştir.

Sonuç olarak, teorik bilgi ve pratik uygulamayı birleştiren bu proje, veri yapılarını gerçek dünyadan bir problem üzerinde kullanma ve bunları etkili bir şekilde görselleştirme yeteneğini geliştirmiştir.

II.Giriş

Bu proje, akademik bir veri seti kullanarak yazarlar

arasındaki işbirliklerini modellemek, analiz etmek ve görselleştirmek amacıyla geliştirilmiştir. Aşağıda proje kapsamındaki temel işlevler özetlenmiştir:

1. **Graf Oluşturma ve İşleme**:
    a. `graf_olustur`: Veri setinden yazarları ve işbirliklerini temsil eden düğümler ve kenarlar oluşturur.
    b. `dugum_ozellikleri_hesabi`: Yazarların makale sayısına göre düğüm boyutu ve renk özelliklerini belirler.

2. **Görselleştirme**:

`pyvis_gorsellestirme`: Pyvis ile grafı HTML formatında görselleştirir.
`altgrafiGorsellestir`: Belirli bir yazar ve işbirlikçileri için alt graf oluşturur ve görselleştirir.
`grafik_kaydet`: Görselleştirme sonuçlarını dosyaya kaydeder.

3. **Kısa ve Uzun Yol Hesaplama**:

`en_kisa_yol`: Dijkstra algoritması ile iki düğüm arasındaki en kısa yolu hesaplar.
`tumEnKisaYollariHesapla`: Alt graf içindeki tüm düğümler için kısa yolları hesaplar.
`enuzunyolHesapla`: **DFS (Depth-First Search)** algoritmasını kullanarak, belirli bir düğümden başlayarak en uzun yolu hesaplar.

4. **İşbirliği Analizleri**:

`limitliIsbirlikHesapla`: Belirli bir limitte, düğümlerin işbirliği yaptığı yazar sayılarını hesaplar.
`en_fazla_ortak_yazara_sahip`: En çok işbirliği yapan yazarı ve bağlantı sayısını belirler.
`isbirlikKuyruguOlustur`: Belirli bir yazarın işbirlikçilerini sıralayarak görselleştirir.

5. **BST (Binary Search Tree) İşlemleri**:

`BSTolusturVeGorsellestir`: En kısa yol verilerini kullanarak bir ikili arama ağacı oluşturur, bir düğümü siler ve sonuçları görselleştirir.

6. **Dinamik HTML ve Webview Arayüzü**:

Proje sonuçları, kullanıcıya özel HTML ve JavaScript tabanlı bir arayüzle sunulmaktadır. Kullanıcılar bu arayüzden analizleri çalıştırabilir ve sonuçları gözlemleyebilir.

III. Yöntem

Bu projede yazarlar arasındaki işbirliği ilişkilerini analiz etmek için iki temel algoritma kullanılmıştır: **Dijkstra Algoritması** ve **Derinlik Öncelikli Arama (DFS)**. Bu yöntemler, graf üzerindeki yolların özelliklerini incelemek ve işbirliği ağlarını anlamak için uygulanmıştır.

Dijkstra Algoritması (En Kısa Yol)

Dijkstra algoritması, bir graf üzerinde bir başlangıç düğümünden diğer düğümlere olan en kısa yolları bulmak için kullanılır. Bu, projede yazarlar arasındaki en kısa işbirliği yolunu hesaplamak amacıyla uygulanmıştır.

1. **Nasıl Çalışır?**

Her düğümün başlangıçta sonsuz mesafede olduğu varsayılır, ancak başlangıç düğümünün mesafesi sıfırdır.
İşlenecek düğümler bir kuyrukta saklanır ve her adımda en düşük mesafeye sahip düğüm işlenir.
Komşu düğümler ziyaret edilerek mesafeler güncellenir; daha kısa bir yol bulunduğunda mesafe ve önceki düğüm bilgileri değiştirilir.

2. **Sonuç:**

Kaynak düğümden hedef düğüme kadar olan en kısa yol ve bu yolun toplam ağırlığı (örneğin, makale sayısına dayalı) hesaplanır.
Bu algoritma, yazarlar arasındaki en verimli işbirliği yollarını belirlemek için kullanılmıştır.

Derinlik Öncelikli Arama (DFS - En Uzun Yol)

Derinlik Öncelikli Arama (DFS), graf üzerinde mümkün olduğunca derine inerek düğümleri ziyaret eden bir algoritmadır. Bu, projede belirli bir yazarla

başlayarak en uzun işbirliği yolunu bulmak için uygulanmıştır.

1. **Nasıl Çalışır?**

   Başlangıç düğümünden itibaren, ziyaret edilmemiş komşulara doğru ilerlenir.
   Her bir yolda derinlemesine arama yapılır ve daha uzun bir yol bulunduğunda bu yol kaydedilir.
   Ziyaret edilen düğümler tekrar işlenmez, böylece döngülerden kaçınılır.
   DFS, LIFO (Last In, First Out) prensibi ile çalışır. Bu, derinlemesine aramayı sağlamak için yığın (stack) veri yapısının kullanılmasını gerektirir.

2. **Sonuç:**

   Başlangıç düğümünden başlayarak en uzun bağlantı zinciri (yazarlar arasındaki en geniş işbirliği ağı) hesaplanır.
   Bu yöntem, graf üzerindeki tüm potansiyel işbirlikçi yolları keşfetmek için kullanılmıştır.

IV. Deneysel Sonuçlar

Bu proje, **Anaconda dağıtımında Spyder IDE** kullanılarak geliştirilmiştir. **Anaconda**, veri analizi ve bilimsel hesaplamalar için önceden yapılandırılmış Python ortamı sunması nedeniyle tercih edilmiştir. İçerisinde bulunan **Spyder IDE**, kullanıcı dostu bir arayüz ve hata ayıklama araçları ile proje geliştirme sürecini kolaylaştırmıştır. Ancak, Python'un performans sınırlamaları ve görselleştirme işlemleri sırasında bazı zorluklarla karşılaşılmıştır.

1. Python'un Performansı

Python'un yorumlanan bir dil olması, özellikle büyük veri kümeleri ve real-time işlemlerde **yavaşlık** ve **hantallık** sorunlarına neden olmuştur. Bazı işlemler beklenenden uzun sürmüş ve bu durum kullanıcı deneyimini olumsuz etkilemiştir.

2. Görselleştirme Sorunları

Başlangıçta görselleştirme sonuçları HTML dosyası olarak tarayıcıda açılmıştır. Ancak:

- **Tarayıcıların tek çekirdek kullanması**, uzun bekleme sürelerine neden olmuştur.
- Büyük graf dosyalarında **tarayıcı çökmeleri** yaşanmıştır.

Daha sonra, **Webview API** ile masaüstü uygulamasına geçiş yapılmıştır. Bu yöntem, görselleştirme işlemlerini daha hızlı ve stabil hale getirerek sorunu çözmüştür.

3. Flask API'nin Kullanılamaması

Flask API entegrasyonu planlanmış ancak:

- **Zorlu yapılandırma**,
- Tarayıcı tabanlı real-time veri işleme uyumsuzluğu nedeniyle uygulanamamıştır.

Bunun yerine, **Webview API** daha kolay bir çözüm sunarak görselleştirmeyi masaüstünde hızlı ve etkili bir şekilde gerçekleştirmiştir.

V.Sonuçlar

Bu proje, akademik bir veri seti kullanılarak yazarlar arasındaki iş birliği ilişkilerini modelleyen bir graf yapısı oluşturmayı ve bu yapı üzerinde çeşitli veri yapısı ve algoritma konseptlerini uygulamayı amaçlamıştır. Proje kapsamında hem teorik bilgi hem de uygulamalı becerilerin geliştirilmesi hedeflenmiştir.

Proje, Excel dosyalarından çekilen verilerin işlenmesiyle başlamış, Python'un güçlü kütüphanelerinden **Pandas** ve **Pyvis** kullanılarak grafik modelleme ve görselleştirme gerçekleştirilmiştir.

Son olarak, aşağıdaki bölümlerde kullanılan algoritmaların temel işleyişine dair kaba kod parçaları sunulacaktır. Bu kodlar, algoritmaların nasıl çalıştığını daha iyi anlamaya yardımcı olacaktır. Proje, teorik bilgi ve pratik uygulamaların başarılı bir birleşimi olarak tamamlanmıştır.

VI. Algortimanın Kaba Kodu:

En Kısa Yol:

```
FONKSİYON en_kisa_yol(graph, start, end)
  mesafeler = Tüm düğümler için sonsuz mesafe ata
  mesafeler[start] = 0
   onceki_dugumler = Tüm düğümler için 'None' ata
   kuyruk = [(start, 0)]
```

```
WHILE kuyruk boş değilse
    // En küçük mesafeye sahip düğümü seç
    kuyruk'u mesafeye göre sırala
    (mevcut_dugum, mevcut_mesafe) = kuyruk'tan ilk elemanı çıkar
    IF mevcut_mesafe > mesafeler[mevcut_dugum] THEN
        CONTINUE
    // Mevcut düğümün tüm komşularını ziyaret et
    FOR HER komsu, agirlik graph.komsu_listesi[mevcut_dugum] DO
        // Yeni mesafeyi hesapla
        mesafe = mevcut_mesafe + agirlik
        // Eğer yeni mesafe daha küçükse, güncelle
        IF mesafe < mesafeler[komsu] THEN
            mesafeler[komsu] = mesafe
            onceki_dugumler[komsu] = mevcut_dugum
            kuyruk'a (komsu, mesafe) ekle
        ENDIF
    ENDFOR
ENDWHILE

// En kısa yol ve mesafeleri döndür
RETURN mesafeler, onceki_dugumler
ENDFONKSİYON
```

En Uzun Yol:

```
en_uzun_yol_hesapla(graph, baslangic_id)
// Eğer başlangıç düğümü graf içinde yoksa, None döndür
IF baslangic_id graph.komsu_listesi içinde DEĞİLSE THEN
    RETURN None
ENDIF
// Derinlemesine arama (DFS) için iç içe fonksiyon
FONKSİYON dfs(node, bakilanlar)  // Mevcut düğümü bakılanlar kümesine ekle
    bakilanlar'e node ekle
    max_path = []  // En uzun yol başlangıçta boş
    // Mevcut düğümün tüm komşularını dolaş
    FOR HER yan_dugum, agirlik IN graph.komsu_listesi[node] DO
        // Eğer komşu düğüm henüz bakılmadıysa
        IF yan_dugum bakilanlar içinde DEĞİLSE THEN
            // Komşu düğümden başlayarak DFS yap
            path = dfs(yan_dugum, bakilanlar)  Eğer bulunan yol mevcut maksimum yoldan uzunsa,
            IF UZUNLUK(path) > UZUNLUK(max_path) THEN
                max_path = path
            ENDIF
        ENDIF
    ENDFOR
    // Mevcut düğümü bakılanlar kümesinden çıkar
    bakilanlar'den node çıkar
    RETURN [node] + max_path  // Mevcut düğümü yolun başına ekle
ENDFONKSİYON

// Başlangıç noktası için DFS başlat
bakilanlar = BOŞ KÜME
RETURN dfs(baslangic_id, bakilanlar)
ENDFONKSİYON
```

VII. Yazar Katkıları

Abdullah Önder: Projenin graf yapısını oluşturma kısmını gerçekleştirmiştir. Yazarlar ve iş birliği ilişkilerini modelleyen düğüm ve kenar yapılarını geliştirmiştir.

Sadık Gölpek: Projede yer alan isterlerin uygulanmasını sağlamıştır. Algoritmaların geliştirilmesi, graf analizleri ve görselleştirme süreçlerini yürütmüştür.

Görselleştirme ve diğer işlemler ortak çalışma sonucunda yapılmıştır.

VIII. Kaynakça:

Python dilini öğrenirken kaynak:

https://www.youtube.com/+Sadievrenseker_BK

Veri yapıları ve Algortimaları öğrenirken kaynak:

https://bilgisayarkavramlari.com/

ChatGPT. Aldığımız bazı hatalarının çözümü için kullanılan bilgiler:

https://chatgpt.com/


Başlangıç
Veri Okuma (read_excel())
Graf Yapısını Oluştur (graf_olustur())
Düğüm Özelliklerini Hesapla (dugum_ozellikleri_hesabi())
Düğümleri ve Kenarları Ekle
Grafı Görselleştir ve Kaydet (pyvis_gorsellestirme())
HTML Arayüzü Oluştur
Kullanıcı İşlemleri
En Kısa Yol Hesaplama ve Görselleştirme (enkisayolHesapla())
En Uzun Yol Hesaplama (enuzunyolHesapla())
İşbirlikçi Kuyruğu Oluşturma (isbirlikKuyruguOlustur())
BST Oluşturma ve Düğüm Silme (BSTolusturVeGorsellestir())
Alt Graf ve Kısa Yolları Hesaplama (isbirlikYollariHesapla())
En Fazla İşbirliği Yapan Yazar Bulma (en_fazla_ortak_yazara_sahip())
İşbirliği Yapılan Yazar Sayısı Hesaplama (limitliIsbirlikHesapla())
Sonuçları Görselleştir ve Kaydet
Çıkış