PROGRAMLAMA LABORATUVARI
YAZARLAR VE İŞ BİRLİĞİ ANALİZİ

Cemile AKSU
Kocaeli Üniversitesi
Mühendislik Fakültesi
Bilgisayar Mühendisliği
230201022@kocaeli.edu.tr

Leyla Tuana
BOZANER
Kocaeli Üniversitesi
Mühendislik Fakültesi
Bilgisayar Mühendisliği
230201066@kocaeli.edu.tr

Özet: Bu projede nesneye yönelik programlama dillerinden biri olan Python ile akademik bir veri seti kullanılarak yazarlar arasındaki iş birliği ilişkilerini modelleyen bir graf yapısı oluşturmayı ve bu graf üzerinden çeşitli veri yapısı ve algoritma konseptlerini uygulamayı amaçladık. Proje, düğümler olarak yazarları, kenarlar olarak yazarlar arasındaki ortak makale ilişkilerini temsil eden bir grafik yapı geliştirmeyi hedeflemiştir. Bu kapsamda veri seti işlenmiş, grafik modelleme gerçekleştirilmiş ve analiz sonuçları yorumlanmıştır.

I. GİRİŞ

Proje kapsamında, akademik bir veri seti kullanılarak yazarlar arasındaki iş birliği graf ile modellenmiş ve görselleştirilmiş, oluşturulan model üzerinde farklı algoritmalar uygulanmış ve çeşitli analizler gerçekleştirilmiştir. Amacımız hem teorik hem de uygulamalı becerilerimizi geliştirmek ve bu süreçte graf veri yapısı, algoritma tasarımı ve görselleştirme araçlarını kullanmaktır.

Projenin içeriği, nesneye yönelik programlama temelleri ile yazılmaya uygun bir projedir. Çok platformlu bir yapıya sahip olan, düzenleme, hata ayıklama, test yönetimi süreçlerini destekleyen, nesneye yönelik programlama temellerini hızlı kavratan ve birçok pozitif özelliğe sahip olan Python dili ile kodlanmıştır. Python, basit ve okunabilir bir sözdizimine sahiptir. Web geliştirme, veri analitiği, yapay zeka, bilimsel hesaplama ve grafiksel görselleştirme gibi birçok alanda kullanılabilir. Matematiksel işlemler, veri analizi, grafik oluşturma ve ağ analizi için güçlü kütüphaneler sunar. Projemizde yazarlar arasındaki iş birliklerini modellemek ve görselleştirmek gibi grafiksel ve algoritmik işlemler yer alıyor. Bu işlemler, güçlü yapıya sahip olması nedeniyle Python ile gerçekleştirilmiştir.

II. YÖNTEMLER ve TASARIM

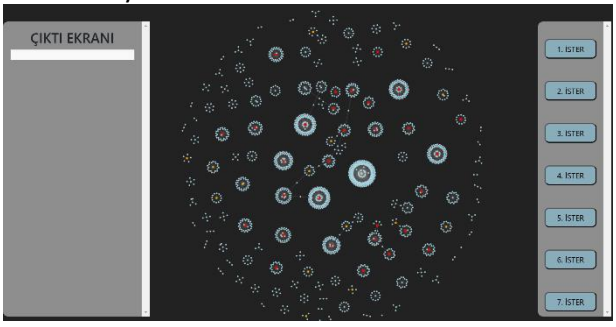
Bu bölümde modellediğimiz graf yapısını oluşturmak için kullanılan yöntemler üzerinde durulacak ve detaylı bilgilendirme yapılacaktır.

A. Arayüz Tasarımı

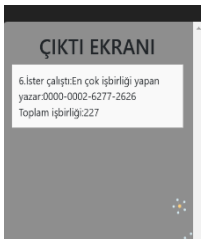
Projemizin arayüz ekranını tasarlamak için öncelikle excel dosyalarını okumada ve verileri analiz etmede oldukça faydalı olan **Pandas** kütüphanesi, web framework olan **Flask**, grafik veri görselleştirme desteği sunan **PyVis** kütüphanesi, arayüzün renk düzenlemeleri ve yerleşimi için **HTML ve CSS**, kullanıcı etkileşimlerini ve dinamik veri işleme süreçlerini yönetmek için **JavaScript** kullanılmış, özellikle belirli API uç noktalarına çağrılar yapılması ve canlı güncellemelerin gösterilmesi sağlanmıştır. **NetworkX,** aynı şekilde düğümlerin ve kenarların tanımlanmasında, en kısa yol-en uzun yol ve graf oluşturma gibi algoritmalarda kullanılmıştır.
PyVis ile oluşturulan etkileşimli grafikler HTML formatında sunulmuştur. Flask, bu grafikleri tarayıcıda gösterir ve kullanıcıların dinamik bir şekilde grafiği incelemesine olanak tanır.

Arayüz Bileşenleri:

- **Ana Görsel Alan :** PyVis ile oluşturulan dinamik grafiklerin gösterildiği bir alan. Kullanıcılar düğümler yardımıyla hem author hem coauthor yazarlar hakkında bilgi alabilirler ve ortak makale sayılarını temsil eden kenar ilişkilerini görsel olarak inceleyebilirler.

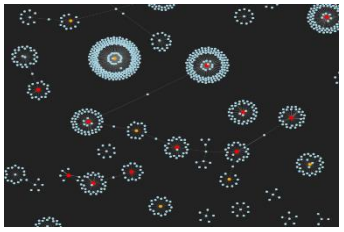


- **Çıktı Ekranı:** İşlemlerin sonuçlarını ve API çağrılarından dönen yanıtları görüntülemek için kullanılan alan. Bu ekranda; işlem adımları, sonuçlar ve hata mesajları detaylı şekilde kullanıcıya sunulur.



- **Buton Ekranı:** Kullanıcının API çağrılarını kolayca yapabileceği butonlar ve veri giriş kutuları (text-box) bulunur. Bu kullanıcı, isterlerin sonuçlarına butonlar yardımıyla ulaşabilir, ID girişlerini gerçekleştirebilir.

- **Renk Kodlaması:** Veri setinde bulunan tüm yazarların yazdığı makale sayılarına göre ortalamanın yüzde 20 üzerinde olan düğümler belirli olacak şekilde daha büyük ve renk olarak kırmızı , yüzde 20 altında olacak düğümler daha küçük dairelerle ve daha açık renk olacak şekilde açık mavi ile gösterilmiştir. Yüzde 30 olanlar da ekstra olacak şekilde turuncu ile gösterilmiştir.



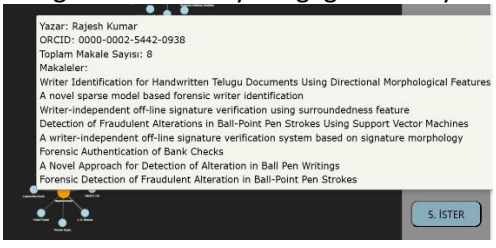
Kullanıcı Etkileşimleri:

- **API Tetikleme (Flask Kullanımı):** Kullanıcılar, ister butonu ekranında butonlar yardımıyla belirli API işlemlerini başlatabilir. Örneğin ister1 için POST metodunu kullanarak backend dahilinde haberleşmeyi gerçekleştirdik.

```
@app.route( rule: "/api/ister1", methods=['POST'])
def ister1_basla():
    global olusan_graf, ister1_sirala, ister1_sonuc
    veri = request.get_json()
    id1 = veri.get('id1')
    id2 = veri.get('id2')
```

/api/ister1/start, /api/ister2, /api/ister3 gibi URL'ler, belirli işlemleri başlatmak için API endpoint'leri olarak tanımlanmıştır.
@app.route("/") ile bir web sayfası kullanıcıya sunulmaktadır. Bu sayfa, PyVis ile oluşturulan grafikleri ve JavaScript tabanlı etkileşimleri içermektedir.
Thread kullanımı ile API çağrıları sonucu başlatılan işlemler, arka planda çalıştırılmaktadır. Böylece kullanıcıya işlem sırasında adım adım bilgi verilmektedir. Aynı zamanda **Threading**, uzun süreli işlemleri (örneğin, en kısa yol hesaplama) ana işlem döngüsünü bloklamadan gerçekleştirmek için kullanılmıştır. Bu, kullanıcının diğer işlemleri aynı anda yapabilmesine olanak tanır. PyVis ile oluşturulan görselleştirme, Flask üzerinden HTML formatında kullanıcıya gönderilmektedir. Flask, dinamik verileri alıp bu verilerden grafik oluşturup kullanıcıya iletilmesini sağlamaktadır. Flask, bu şekilde hem bir web sunucusunu çalıştırıyor hem de bir API sağlayıcı olarak haberleşmede görev almaktadır.

- **Dinamik Görselleştirme:** Kullanıcılar grafik üzerinde düğümlere tıklayarak düğüme dair detaylı bilgi görüntüleyebilir.



Örneğin, author düğümünün üzerine tıkladığımızda ekranda oluşan görüntü

- **Canlı Güncellemeler:** İşlem devam ederken adım adım güncellemeler çıktı ekranında gösterilir.

Örneğin, ister2 sonucu.

B. Proje Mantığı ve Kullanılan Temel Veri Yapıları

Bu bölümde proje kapsamında kullanılan sınıflar ve projenin temel mantığında kullanılan yaklaşımlar detaylıca ele alınacaktır.

Graf veri yapısı, temelde düğüm (node) ve kenar (edge) gibi unsurlardan oluşur. Bu projenin kapsamında, yazarların düğümleri (nodes); yazarlar arasındaki bulunan iş birliklerinin kenarları (edges) temsil ettiği bir Graf veri yapısı oluşturulmuştur.

Graflar üzerinde çeşitli veri yapısı kavramlarının uygulanması, hem pratik hem de teorik olarak oldukça önemlidir. Proje dahilinde arama (search), sıralama (sort), alt graf oluşturma (subgraph creation), en kısa yol ve en uzun yol bulma gibi işlemler yapılmıştır. Bu işlemler sayesinde, proje boyunca veri yapılarının daha iyi anlaşılması ve algoritmaların etkili bir biçimde kullanılması sağlanmıştır.
Proje, akademik iş birlikleri gibi somut ve günlük hayatta karşılaşılabilecek bir problem üzerinden veri yapıları ve algoritmaların uygulanmasını sağlamaktadır. Yazarların iş birliği yaptığı ortak makaleler üzerinden bir graf oluşturularak, akademik topluluklar arasındaki etkileşimlerin derin bir analizi yapılmıştır. Bu sayede, farklı disiplinler arasındaki iş birliklerinin yoğunluğunu anlamak ve etkisi yüksek yazarları tespit etmek gibi pratik çıkarımlar yapılmasını sağlar.

Graf yapısının görselleştirilmesi, proje kapsamınca yapılan analizlerin daha etkili bir şekilde sunulmasını sağlamıştır. Projede, grafiksel temsil araçlarını kullanarak, yazarlar arasında bulunan ilişkilerin görsel bir modeli oluşturulmuştur. Görselleştirme sayesinde karmaşık bir veri yapısı olan grafların anlaşılması kolaylaştırılmış; süreç boyunca kullanılan araçlar ile düğüm büyüklüğünün ve renginin makale sayısına göre değiştirilmesi ile yazarlar arası ilişkilerin okunabilirliği kolaylaştırılmıştır.

C. Kullanılan Algoritmalar ve Class Yapıları

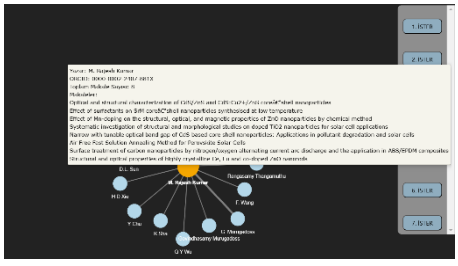
Program, Python programlama dilinde yazılmış olup, geliştirme ortamı olarak PyCharm kullanılmıştır. Python; nesneye yönelik programlama temelleri ve bununla birlikte graf yapısını oluşturmada kullanılan programlama dillerinden bir tanesi olduğu için bu bölümde, projede kullanılan Graf veri yapısı ve kullanılan algoritmalardan detaylıca bahsedilecektir.

Proje için kullanılan veri seti, yazarların adları ve ortak yazdıkları makaleleri içermektedir. Veriler; bir Excel dosyasında düzenlenmiş, her yazar bir düğüm (node) her ortak makale ilişkisi ise bir kenar (edge) olarak modellenmiştir. Kenarlar ağırlıklı olup, iki yazar arasında bulunan ortak makale sayısını temsil etmektedirler.

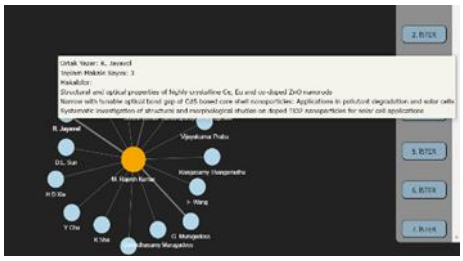
- **GRAF MODELİ**

  Her yazar bir düğüm (node) olarak temsil edilmiştir. Düğümlerde yazar ismi, eğer var ise orcid bilgisi, yazarın yazdığı toplam makale sayısı ve yazılan makalelerin isimlerinin bilgisi bulunmaktadır. Yazarlar arasındaki bağlantı ise kenar (edge) ile temsil edilmiştir. Kenarlarda iki yazar arasındaki ortak makale sayısı ve bu makalelerin isimleri bulunmaktadır.

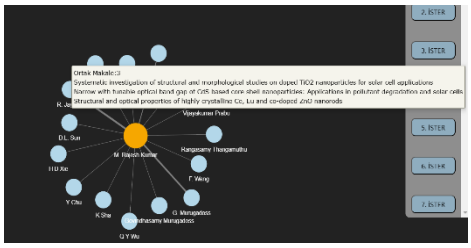
  Grafta bulunan düğümlerin boyutları ve renkleri yazarların makale sayılarına göre farklılık göstermektedir. Eğer bir yazarın yazdığı makale sayısı, ortalama makale sayısının %20 'sinin üstünde ise daha büyük ve kırmızı renkte; eğer %20 civarında ise ortalama boyutta ve turuncu renkte; %20'nin altında ise daha küçük ve mavi renkte gösterilmiştir.



Author düğümünün üzerine tıklandığında oluşan görüntü



Coauthor düğümünün üzerine tıklandığında oluşan görüntü



İki yazar arasında bulunan kenar üzerine tıklandığında oluşan görüntü

- **Breadth-First Search (BFS) Algoritması**

  BFS algoritması, iki düğüm arasındaki en kısa yolu hesaplamak için kullanılır. Projede bulunan 1. İster için yazılmış IkiDugum_MesafeHesapla(id1,id2): adlı fonksiyonda kullanıcıdan iki adet orcid id'si alarak bu iki düğüm arasında bulunan en kısa mesafeyi ve iki düğüm arasındaki yol boyunca hangi düğümler ziyaret ediliyor ise onları hesaplamayı sağlar.

  Bu hesaplamayı yaparken öncelikle deque kullanarak bir FIFO (First In First Out) kuyruğu oluşturulmuştur ve başlangıç değeri olarak id1 değeri kuyruğa eklenmiştir. Fonksiyon boyunca ziyaret edilen tüm düğümleri takip etmek için tamamlanan adlı bir set veri yapısı oluşturulmuştur. Bu sayede aynı düğümün birden fazla kez ziyaret edilmesi engellenmiştir. Ayrıca her adımda kuyruğa hangi düğümlerin eklendiği adım adım yazılmıştır ve kullanıcıya gösterilmiştir. Ayrıca iki yazar arasında oluşacak olan en kısa mesafe hesaplanmış ve fonksiyonun sonucunda; iki yazar arasındaki en kısa yol boyunca uğranan tüm düğümler gösterilmiştir.

- **Binary Search Tree (BST)**

  BST yapısı, bir kuyruk, liste vb. yapıların içinde yapılan hesaplama, sıralama gibi işlemleri yapmamızda kolaylık sağlar. 3. İster için 1. İster sonucunda oluşa kuyruk yapısını bir BST yapısına çevirmek için Binary_Tree adlı bir class oluşturulmuştur. Bu class içerinde oluşan düğümlerin tepe düğüme (root) göre daha küçük olan değerlerin solda, daha büyük olanların ise sağ tarafta olacak şekilde ağaç yapısı içine eklemeyi sağlayan fonksiyonlar oluşturulmuştur. BinaryTree_Olustur(key, id_kaldir): adlı fonksiyonda ise öncelikle 1. İster sonucunda oluşan kuyruk yapısı alınarak bu kuyruk yapısını bir BST yapısına çevirmek için Binary_Tree adlı class nesnesi kullanılır. Kuyrukta bulunan bilgileri bu nesneye aktararak BST yapısını oluşturmaktadır. Sonrasında oluşturulan bu ağaçtan girilen id değerine sahip olan düğüm kaldırılır ve BST yapısının son hali kullanıcıya gösterilir.

- **Djikstra Algoritması**

  Djikstra algoritması, iki düğüm arasında bulunan en kısa yolu hesaplamak için kullanılan bir algoritmadır. Projede bulunan 4. İster için yazılmış EnKisaYol_Hesapla(graf, baslangic_dügümü): bu fonksiyonda; djikstra algoritmasının temelleri esas alınarak bir yazarın, diğer yazarlarla olan ilişkilerinin ağırlıklı kenarlarda bulunan değerlere göre en kısa yollar hesaplanmıştır.

  Bu hesaplamayı yaparken mesafe = {} adlı bir sözlük veri yapısında değişken ile iki yazar arasında bulunan kenar ve bu kenarın ağırlık değeri anahtar-değer çifti olarak tutulmuştur. Sonrasında ilgili yazarın kendisi dışındaki tüm yazarlar ile arasında mesafe başlangıçta sonsuz olarak belirlenmiştir. Eğer yazarın diğer yazarlardan herhangi biriyle ilişkisi bulunuyor ise iki yazar arasındaki mesafe değeri en kısa olacak şekilde güncellenmiştir. Bu fonksiyon ile ilgili yazarın kendisi hariç bağlantılı olduğu tüm yazarlar ve bu yazarlara olan en kısa mesafeleri teker teker hesaplanarak bu değerler bir tabloda gösterilmiştir.

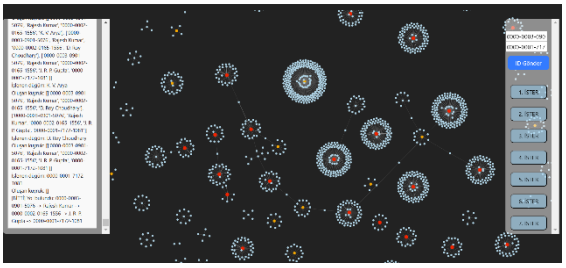
- **Depth First Search (DFS) Algoritması**

  Depth First Search algoritması, graf üzerinde bulunan bir düğümden başlayarak en uzun yolu bulmayı sağlar.

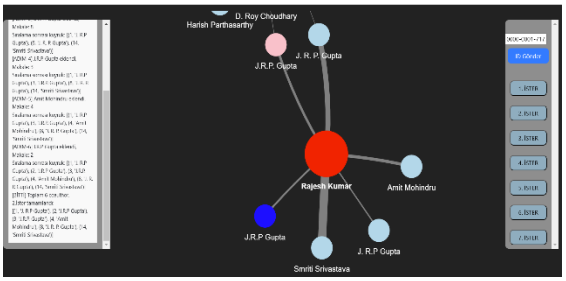
Projede bulunan 7. İster için yazılmış olan EnUzunYol_Hesapla(graf, baslangic_dugumu): fonksiyonu DFS algoritmasını temel alarak, bir yazarın diğer yazarlar arasındaki bağlantıları kontrol ederek bu bağlantılar arasındaki en uzun yolu bulmayı sağlar. En uzun yolu bulurken yazarlar arasındaki kenar ağırlıklarına değil düğüm sayısına bakarak en uzun yolu bulmayı sağlar. Bulunan en uzun yol boyunca uğranan düğümler bir yığın içinde tutulur ve sonrasında oluşan bu yolun son hali kullanıcıya gösterilir.

III. DENEYSEL SONUÇLAR

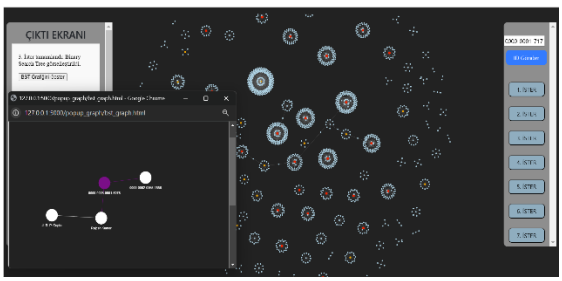
1.İster çalıştırıldığında oluşan sonuç:



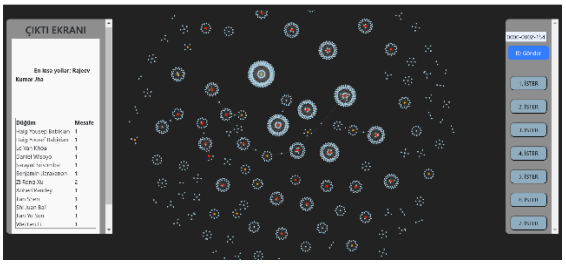
2.İster çalıştırıldığında oluşan sonuç:



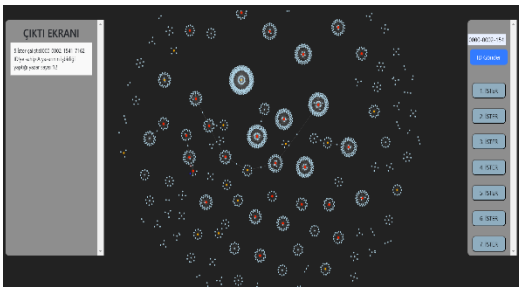
3.İster çalıştırıldığında oluşan sonuç:



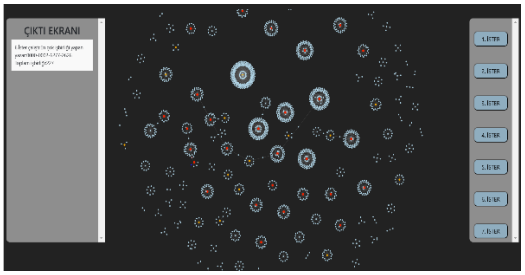
4.İster çalıştırıldığında oluşan sonuç:



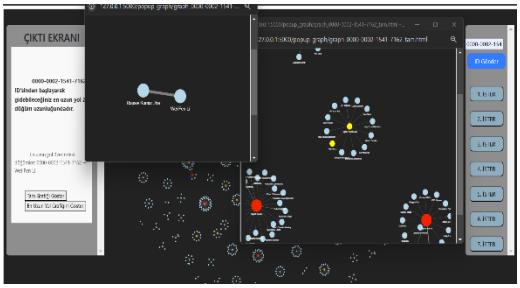
5.İster çalıştırıldığında oluşan sonuç:



6.İster çalıştırıldığında oluşan sonuç:



7.İster çalıştırıldığında oluşan sonuç:

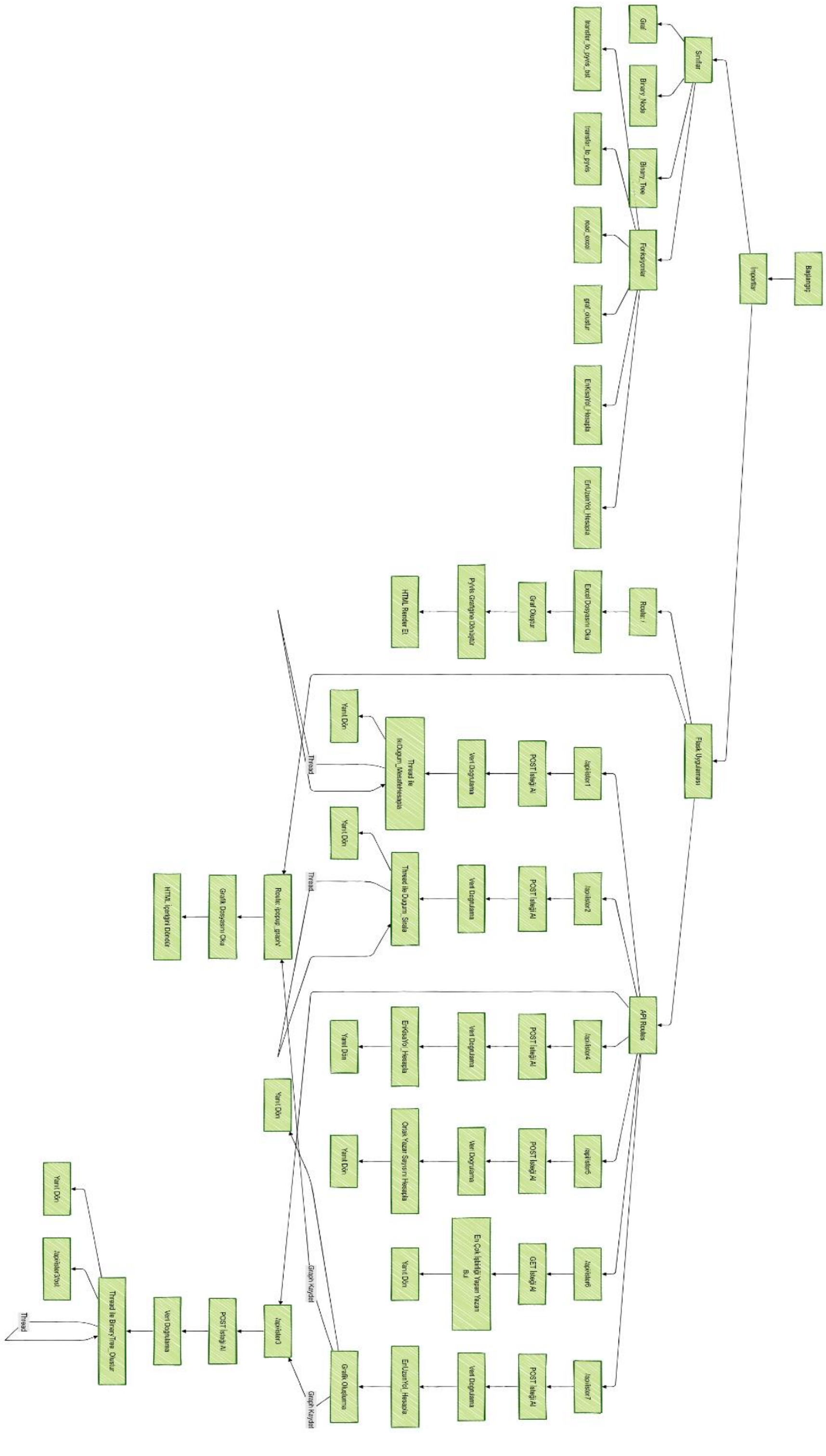


IV. PSEUDO (YALANCI) KOD

```
Gerekli kütüphaneleri içe aktar:
        - pandas, pyvis, flask, collections, ast, os,
time, threading

Flask uygulamasını başlat:
        app = Flask(__name__)

Sınıf Graf:
        Düğümler ve kenarlar için boş sözlükler
oluştur

Fonksiyon add_node():
        Yeni düğümü ekle, eğer mevcut değilse

Fonksiyon add_edge():
        Yeni kenarı ekle, makaleleri ve ağırlığı
güncelle

Sınıf Binary_Node:
        key, sol ve sağ çocukları tanımla

Sınıf Binary_Tree:
        Kök düğümü tanımla
        Fonksiyon insert():
                Anahtarı ekle
        Fonksiyon sil():
                Anahtarı sil
        Fonksiyon sirala():
                In-order gezinti ile düğümleri sırala
```

```
Fonksiyon transfer_to_pyvis_bst(Binary_Tree,
id_kaldir):
	PyVis ağı oluştur
	Ağacı dolaşarak düğümleri ve kenarları
	ekle
	Renkleri belirle (kök, silinen düğüm,
	diğerleri)
	Ağı döndür

Fonksiyon read_excel(excel_dosya_yolu):
	Excel dosyasını pandas DataFrame olarak
	oku ve döndür

Fonksiyon graf_olustur(okunanveri, limit=1000):
	Graf nesnesi oluştur
	Veriyi sınırlı sayıda işleyerek düğümleri
	ve kenarları ekle
	Düğümlerin boyut ve renklerini belirle
	Grafı döndür

Fonksiyon transfer_to_pyvis(olusan_graf
vurgulanan_dugum=[], vurgulanan_kenar=[]):
	PyVis ağı oluştur
	Düğümleri ve kenarları ekle, gerekirse
	vurgula
	Ağı döndür

Fonksiyon EnKisaYol_Hesapla(graf,
baslangic_dugumu):
	BFS kullanarak en kısa yolları hesapla
	Mesafeleri döndür

Fonksiyon EnUzunYol_Hesapla(graf,
baslangic_dugumu):
	DFS kullanarak en uzun yolu bul
	Yolu ve uzunluğunu döndür



	Route "/":
		Fonksiyon gorsel():
			Excel dosyasını oku
			Graf oluştur
			PyVis ile görselleştir
			HTML ve özel script ile
			sayfayı render et

	Route "/popup_graph/<filename>":
		Fonksiyon popup_graph(filename):
			Dosyayı oku ve içeriği
			döndür Eğer dosya yoksa
			hata mesajı göster

	Route "/api/ister1"[POST]:
		Fonksiyon ister1_basla():
			id1 ve id2 al
			Geçerli graf ve ID
			kontrolleri yap
			Eğer uygun ise, BFS ile en
			kısa yolu bulmak için
			thread başlat
			Başlangıç mesajı döndür

	Route "/api/ister1/status"[GET]:
		Fonksiyon ister1_status():
			id1 ve id2 al
			İlgili işlemin durumunu
		döndür

	Route "/api/ister2"[POST]:
		Fonksiyon ister2_basla():
			id1 al Geçerli graf ve ID
			kontrolleri yap
			Eğer uygun ise, komşu
			yazarları sıralamak için
			thread başlat
			Başlangıç mesajı döndür

	Route "/api/ister2/status"[GET]:
		Fonksiyon ister2_status():
			id1 al
			İlgili işlemin durumunu döndür

	Route "/api/ister3"[POST]:
		Fonksiyon ister3_basla():
			id_kaldir al
			Gerekli kontrolleri yap
			Eğer uygun ise, BST oluşturmak
			için thread başlat
			Başlangıç mesajı döndür

	Route "/api/ister3/status"[GET]:
		Fonksiyon ister3_status():
			İlgili işlemin durumunu döndür

	Route "/api/ister3/bst"[GET]:
		Fonksiyon ister3_bst():
			BST grafiğini oluştur ve
			döndür
			Popup butonu ile göster

	Route "/api/ister4"[POST]:
		Fonksiyon ister4():
			id1 al
			En kısa yolları hesapla ve
			sonuçları tablo halinde döndür

	Route "/api/ister5"[POST]:
		Fonksiyon ister5():
			id1 al
			İşbirliği yapan yazar sayısını
			hesapla ve döndür
Route "/api/ister6"[GET]:
	Fonksiyon ister6():
		En çok iş birliği yapan yazarı ve
		sayısını bul
		Sonucu döndür

Route "/api/ister7"[POST]:
	Fonksiyon ister7():
		baslangic_dugumu al
		En uzun yolu bul
		Grafiği vurgula ve sonuçları döndür

Eğer bu dosya ana dosya ise:
	Flask uygulamasını debug modunda çalıştır
```

AKIŞ ŞEMASI:



V. SONUÇ

Projede nesneye yönelik programlamanın sahip olduğu sınıf, graf oluşturma, veri analizi ve arayüz tasarlama yöntemlerini daha iyi kavrayıp onları kullanmayı öğrendik. Aynı zamanda, akademik iş birliklerini modellemek ve analiz etmek için graf veri yapısı ve algoritmalarını nasıl kullanılabileceğimizi öğrendik. Elde ettiğimiz sonuçlar, yazarlar arasındaki ilişkilerin görselleştirilmesi ve analiz edilmesi konusunda faydalı bilgiler sağladı. Gelecekte, daha büyük veri setleriyle çalışmak ve geliştirilmiş algoritmalar kullanarak daha karmaşık analizler yapmamızı bu proje sayesinde mümkün kıldık.

VI. KAYNAKÇA

https://bilgisayarkavramlari.com/

https://www.udemy.com/course/sifirdan-ileri-seviyeye-python/learn/lecture/7585500#overview

https://www.youtube.com/watch?v=AMq0LhzavhU&list=PLKnjBHu2xXNNwV1Twc3UtaMBqGJx3CCrU

https://blog.tomsawyer.com/python-graph-visualization