KOCAELİ ÜNİVERSİTESİ BİLGİSAYAR MÜHENDİSLİĞİ

PROGRAMLAMA LABORATUVARI - I

PROJE 3
YAZARLAR VE İŞ BİRLİĞİ ANALİZİ

Saffet Hakan KOÇAK - 230202058 Yusuf BÜLBÜL - 230202050

I. ÖZET

Bu proje, akademik bir veri seti kullanarak yazarlar arasındaki iş birliği ilişkilerini modellemeyi ve analiz etmeyi amaçlamaktadır. Flask tabanlı bir backend, Cytoscape.js ile desteklenen bir frontend ve çeşitli algoritmalar (örneğin BFS, DFS, Dijkstra) kullanılarak yazarlar ve ortak yazarlar arasındaki graf yapısı üzerinde dinamik çözümler sunulmuştur. Proje, veri yapıları ve algoritmaların entegrasyonu sayesinde kullanıcı dostu bir platform sunarak, akademik iş birliği analizini görselleştirme ve detaylı inceleme imkânı sağlamıştır. İlgili isterler, kullanıcıdan alınan girdiler çerçevesinde interaktif bir arayüz ile gerçekleştirilmiştir.

II. GİRİŞ

Graf veri yapıları, veri analitiği ve ağ teorisi uygulamalarında yaygın olarak kullanılmaktadır. Bu proje, akademik iş birliklerin analiz edilmesini grafik veri yapısı ve algoritmalar aracılığıyla ele almıştır. Flask, Cytoscape.js, ve Python'un güçlü veri işleme olanakları bir arada kullanılarak, hem backend hem de frontend bileşenlerinin verimli bir şekilde çalıştığı bir sistem tasarlanmıştır. Kullanıcılar, grafik verilerini inceleyerek akademik iş birliklerini anlamlandırabilir ve analiz sonuçlarını görselleştirebilir. İsterlerde istenilenlere uygun şekilde çeşitli algoritmalar kullanılarak graf veri yapılarıyla yaptırılması istenilen işlemler gösterilir.

III. YÖNTEM

III.A Veri Seti ve Hazırlık

- Projede kullanılan veri seti, Excel formatında sağlanmış ve Pandas kütüphanesi ile okunmuştur.

- Veriler, üç ana başlık altında işlenmiştir:

- Yazar Adı (author_name): Her bir düğümü temsil eder.

- Ortak Yazarlar (coauthors): Düğümler arasındaki kenarları belirler.

- Makale Başlığı (paper_title): Yazarlar arasındaki ilişkilerin niteliğini temsil eder.

- Veri seti üzerinde tekrar eden ilişkiler filtrelenmiş ve sadece eşsiz bağlantılar graf yapısına eklenmiştir.

III.B Grafik Modeli

- Düğüm Temsilleri: Yazarlar, graf düğümleri olarak modellenmiştir. Her düğüm, ilgili yazara ait makaleler ve kimlik bilgilerini saklar.

- Kenar Temsilleri: Kenarlar, iki yazar arasındaki ortak makale sayısıyla ağırlıklandırılmıştır. Daha fazla makale paylaşan yazarlar arasındaki kenarlar daha kalın ve belirgin çizilmiştir.

• JSON Formatı: Graf verisi, verimli bir şekilde işlenebilmesi için JSON formatında kaydedilmiştir. Bu yapı, Flask uygulamasının kolayca veri sağlamasını ve frontend tarafında kullanılmasını mümkün kılmıştır.

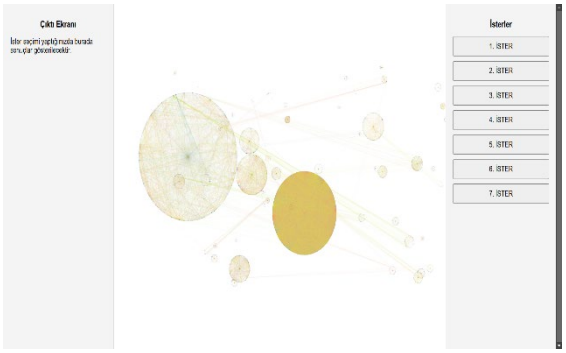

Fig.1. Graf Görseli

III.C Algoritmalar ve Yalancı Kodlar

BFS (Breadth-First Search)

Bu algoritma, bir graf yapısı üzerinde iki düğüm arasındaki en kısa yolun bulunmasında kullanılmıştır. Genişleme bazlı bir yaklaşımla, verilen başlangıç düğümünden hedef düğümüne ulaşılana kadar tüm olası yollar sırasıyla kontrol edilir.

Girdi: Başlangıç düğümü (start), Hedef düğüm (end), Graf (graph)

Çıktı: En kısa yol (en_kisa_yol)

1. Eğer start == end ise

    2. Dönüş [start]

3. Ziyaret edilenler listesi oluştur (visited)

4. Kuyruk oluştur ve [start] ekle

5. Döngü: kuyruk boş değilse

    6. Yolu kuyruktan çıkar

    7. Yoldaki son düğümü bul (node)

    8. Eğer node ziyaret edilmemişse

        9. node'un komşularını kontrol et

        10. Eğer hedef düğüm (end) bulunduysa

            11. Dönüş: Yol + [komşu]

        12. Yeni yolları kuyruğa ekle

        13. node'u ziyaret edilenlere ekle

14. Eğer yol bulunamadıysa

    15. Dönüş: Boş liste

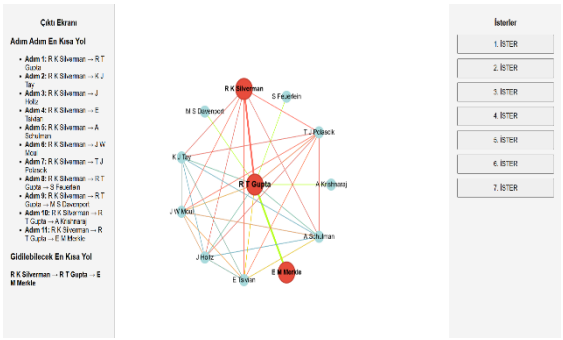

Fig.2. En Kısa Yol

DFS (Depth-First Search)

Bu algoritma, bir düğümden başlayarak ağ üzerindeki maksimum uzunluktaki yolu bulmak için kullanılmıştır. Derinlemesine arama yaparak ağda ilerler ve ziyaret edilmeyen yolları tespit eder.

Girdi: Başlangıç düğümü (start), Graf (graph)

Çıktı: En uzun yol (en_uzun_yol)

1. Ziyaret edilenler listesi oluştur (visited)

2. Geçerli yolu saklamak için bir liste oluştur (current_path)

3. Fonksiyon: DFS(node):

4. node'u visited listesine ekle

5. node'u current_path'e ekle

6. Her komşu için:

7. Eğer komşu ziyaret edilmemişse

8. DFS(komşu) ilerle

9. Geri adım: node'u current_path'den çıkar

10. node'u visited listesinden çıkar

11. DFS(start) çağır

12. En uzun yolu döndür (longest_path)

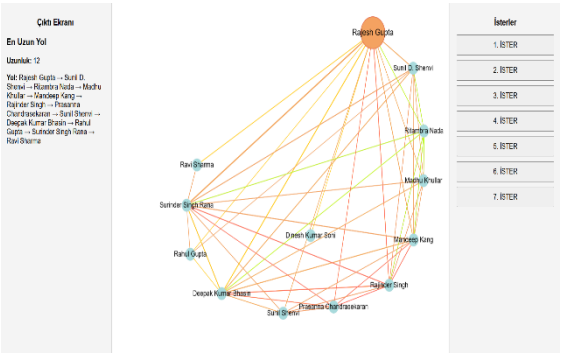

Fig.3. En Uzun Yol

Dijkstra (Shortest Paths)

Dijkstra algoritması, bir graf yapısında verilen bir başlangıç düğümünden diğer tüm düğümlere olan en kısa yolları hesaplamak için kullanılmıştır. Kenar ağırlıkları dikkate alınarak minimum mesafeler hesaplanmıştır.

Girdi: Başlangıç düğümü (start), Graf (graph)

Çıktı: Tüm düğümlere olan en kısa yollar (distances)

1. Tüm düğümler için başlangıç mesafelerini sonsuz yap (distances[node] = inf)

2. Başlangıç düğümü için mesafeyi sıfır yap (distances[start] = 0)

3. Öncelik sırasına göre işlenecek düğümler için bir kuyruk oluştur (queue)

4. Döngü: Kuyruk boş değilse

5. Kuyruktan en düşük mesafeye sahip düğümü çıkar (current)

6. Eğer current daha önce ziyaret edildiyse, devam et

7. current'in tüm komşularını kontrol et:

8. Yeni mesafeyi hesapla (new_distance = distances[current] + edge_weight)

9. Eğer new_distance daha düşükse

10. Mesafeyi güncelle (distances[neighbor] = new_distance)

11. Kuyruğa ekle (queue.push(neighbor))

12. current'i ziyaret edilenlere ekle

13. Tüm mesafeleri döndür (distances)

BST OLUŞTURMA VE GÜNCELLEME

İlk isterde oluşturulan kuyruktaki yazarlardan bir ikili arama ağacı oluşturulmaktadır. Kullanıcıdan bir yazar ID'si istenerek bu yazar ağaçtan çıkarılmıştır ve ağacın son durumu grafiksel olarak gösterilmiştir.

Fonksiyon: yazar_listesi3()

Başla:

1.Veri al

2. author_id ve queue'yu çıkar

3. Eğer queue geçerli değilse:

4. Hata döndür

5. Eğer author_id yoksa:

6. Hata döndür

7. Orijinal kuyruk oluştur

8. Yazarı bul

9. Eğer yazar yoksa:

10. Hata döndür

11. Yazar kuyruktan çıkar

12. Güncellenmiş kuyruk oluştur

13. Yanıt oluştur ve gönder

14. Hata olursa:

15. Genel hata mesajı döndür

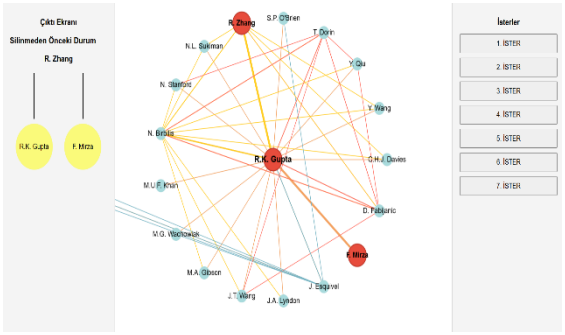

Fig.4. Oluşturulan Binary Search Tree

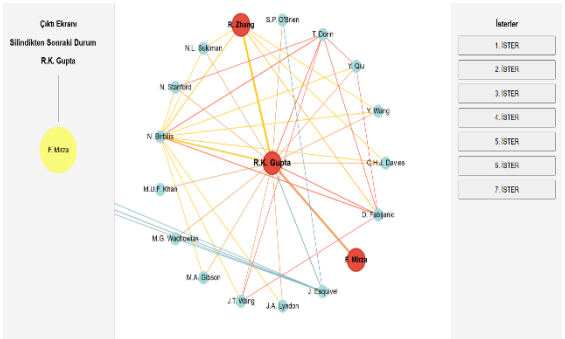

Fig.5. Silinme İşleminden Sonra Binary Search Tree

III.D Flask ve Backend Yapısı

Flask, kullanıcıdan alınan istekleri işleyerek ilgili algoritmayı çalıştırmıştır. Örneğin:

- `/en_kisa_yol` rotası, başlangıç ve bitiş düğümleri arasındaki en kısa yolu hesaplamak için tasarlanmıştır.

- `/kuyruk` rotası, belirli bir yazarın iş birlikçilerini makale sayısına göre sıralar.

- `/en_cok_isbirlik_yapan` rotası, en çok iş birliği yapan yazarı bulur.

- JSON çıktıları, frontend tarafından işlenerek kullanıcıya sunulmuştur.

III.E Frontend ve Kullanıcı Etkileşimi

- Graf Görselleştirme: Cytoscape.js kullanılarak interaktif bir graf oluşturulmuştur. Kullanıcılar düğümlere tıklayarak ilgili düğümün detaylarını sol panelde görebilmektedir.

- İster Seçimi: Sağ paneldeki butonlar, ilgili algoritmayı çalıştırmak için kullanılmıştır. Kullanıcı, belirtilen isterlerden birini seçtiğinde sonuçlar sol panelde görüntülenmektedir.

- Dinamik Vurgulama: En kısa yol veya iş-birlikçilerin vurgulanması gibi işlemler, graf üzerinde görsel olarak belirtilmiştir.

III.F Kuyruk ve Sıralama İşlemleri

- İş birlikçi yazarların sıralanması için özel bir kuyruk yapısı oluşturulmuştur. Makale sayısına göre büyükten küçüğe sıralama yapılmıştır.

• Kuyruğa ekleme ve çıkarma işlemleri adım adım kaydedilmiş ve bu süreçler kullanıcıya görsel olarak sunulmuştur.

III.G Algoritma Akış Diyagramı

• Tüm algoritmaların çalışma prensiplerini göstermek için akış diyagramları hazırlanmıştır. Bu diyagramlar, algoritmaların hangi adımları takip ettiğini açıkça ifade etmektedir. Örneğin:

• BFS algoritması: Başlangıç düğümünden hedef düğüme kadar olan tüm olası yolları kontrol eder.

• DFS algoritması: Derinlemesine arama yaparak maksimum uzunluktaki yolu belirler.

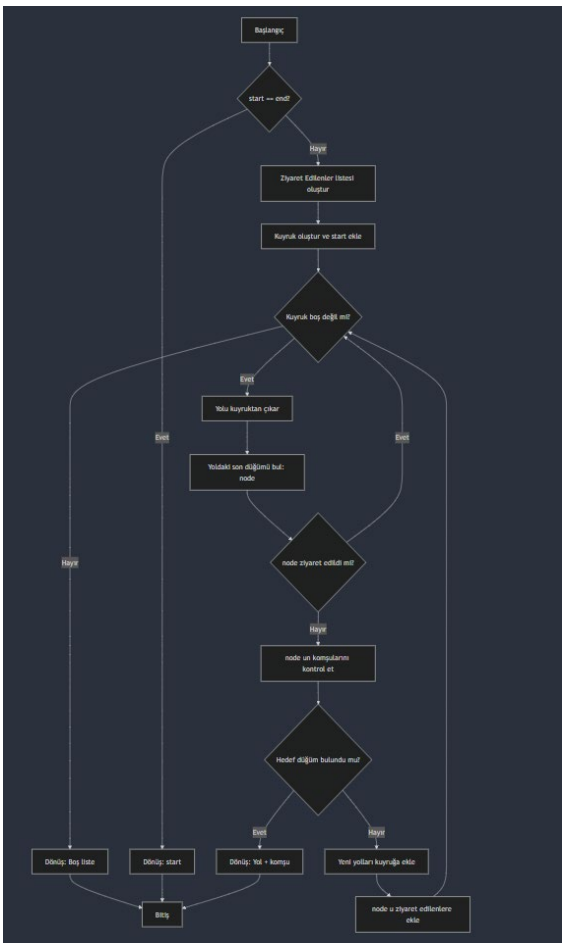

Fig.6. BFS Akış Şeması

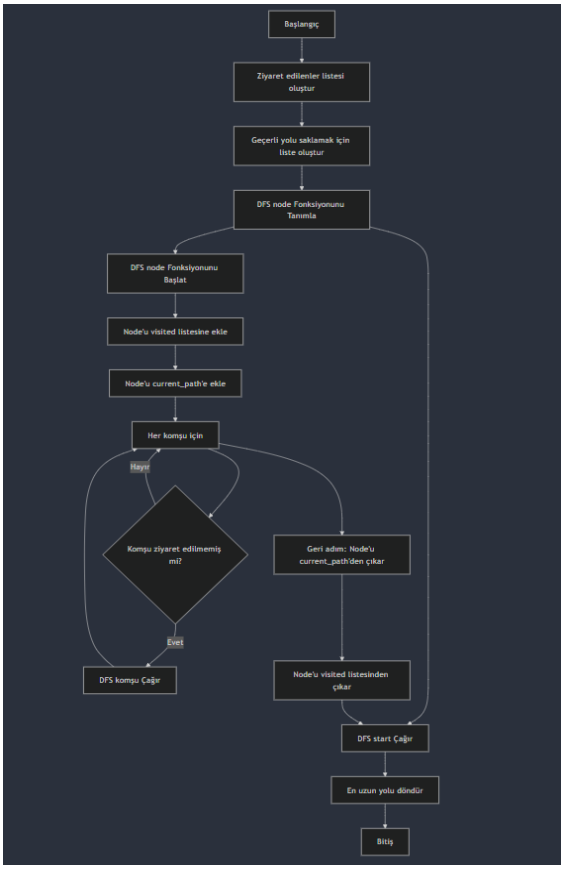

Fig.7. DFS Akış Şeması

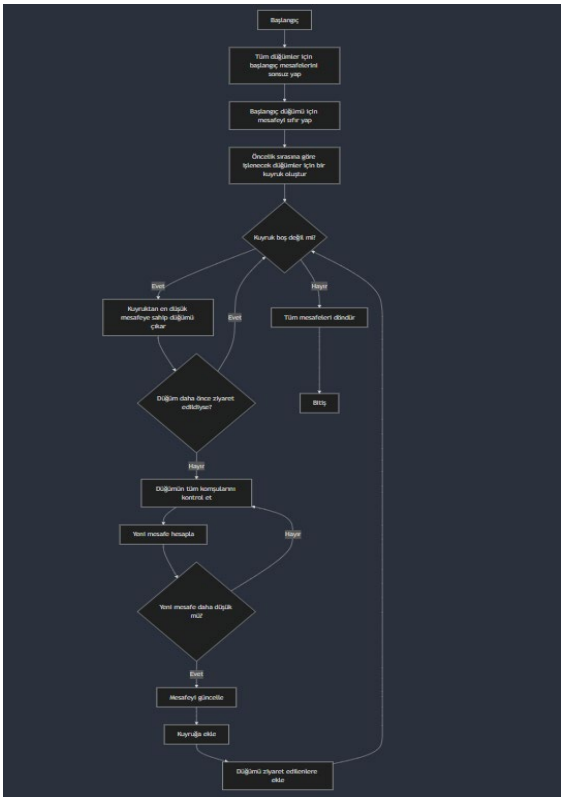

Fig.8. Dijkstra Akış Şeması

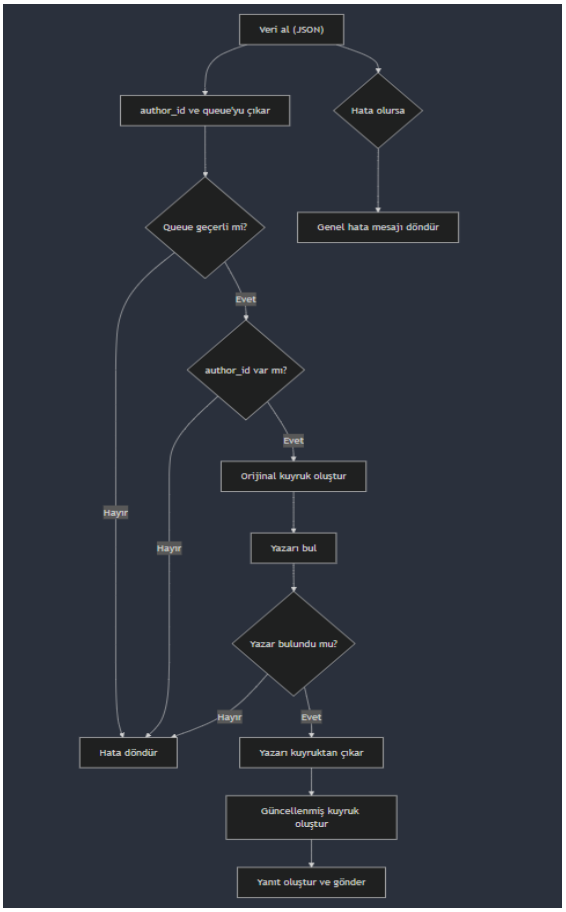

Fig.9. BST Akış Şeması

IV. DENEYSEL SONUÇLAR

IV.A En Kısa Yol Analizi

Kullanıcıdan alınan başlangıç ve bitiş yazar kimlikleri (ID) doğrultusunda, BFS algoritması çalıştırılmıştır. Bu algoritma ile:

• İlgili düğümler ve kenarlar vurgulanarak en kısa yol graf üzerinde işaretlenmiştir.

• Adım adım yapılan güncellemeler sol panelde listelenmiştir. Örneğin:

• Başlangıç düğümünden diğer düğümlere olan bağlantılar kontrol edilmiştir.

• Sırasıyla kenar ağırlıkları dikkate alınarak en kısa yol belirlenmiştir.

IV.B İş Birlikçi Yazarların Kuyrukta Sıralanması

Belirtilen bir yazarın tüm işbirlikçileri, yazdıkları makale sayısına göre sıralanmıştır. Bu işlem sırasında:

• Kuyruğa ekleme ve çıkarma işlemleri adım adım kaydedilmiştir.

• Kullanıcıya sol panelde hem kuyruk güncellemeleri hem de sıralı işbirlikçiler listesi sunulmuştur.

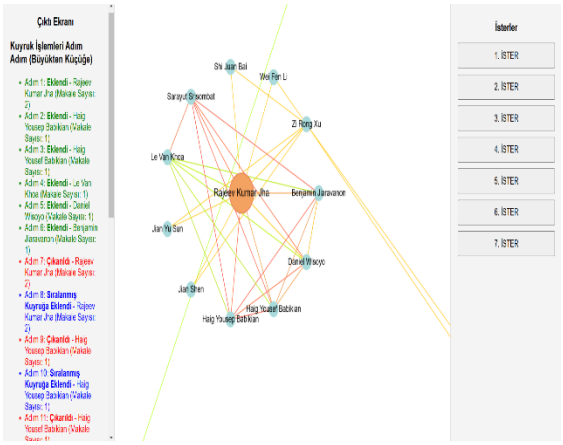

Fig.10. Kuyruğa Ekleme Çıkarma Adımları

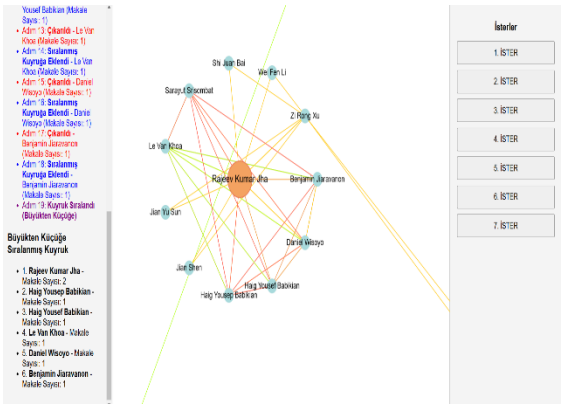

Fig.11. Sıralanmış Kuyruk Gösterimi

IV.C En Uzun Yol Analizi

Bir yazardan başlayarak, iş birlik ağı üzerinde gidilebilecek maksimum yol hesaplanmıştır. Sonuç olarak:

• Ziyaret edilen tüm düğümler ve bağlantılar görselleştirilmiştir.

• Elde edilen yolun toplam uzunluğu sol panelde belirtilmiştir.

IV.D Genel İş Birlik Analizleri

• Toplam İş Birlik Sayısı: Bir yazarın iş birliği yaptığı toplam yazar sayısı hesaplanmıştır.

• En Çok İş Birliği Yapan Yazar: Tüm graf üzerinde en fazla iş birlik yapan yazar belirlenmiş ve bilgisi kullanıcıya sunulmuştur.

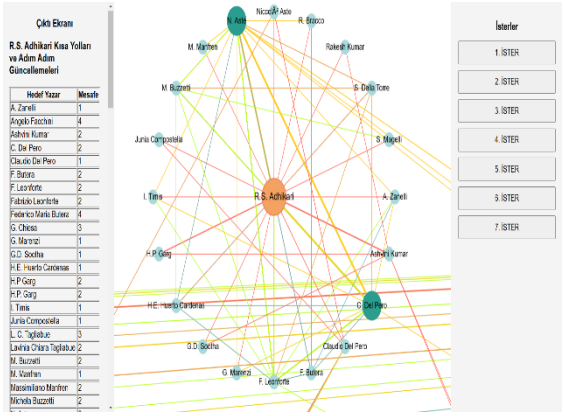

Fig.12. En Kısa Yolların Tabloda Gösterimi

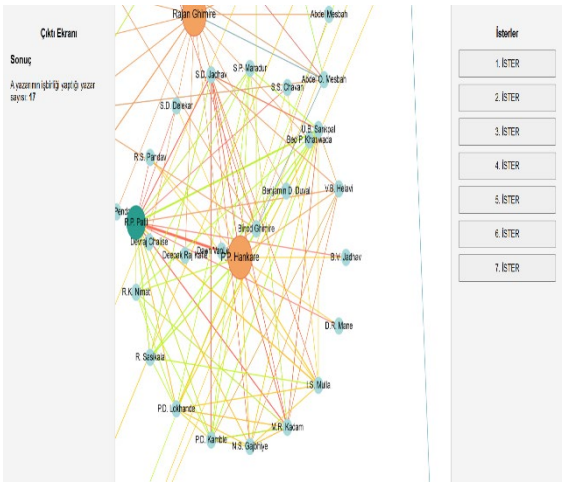

Fig.13. Toplam İş Birlik Sayısı

Fig.14. En Çok İş Birliği Yapan Yazar

V. SONUÇ

Bu proje, veri yapıları ve algoritmaların somut bir problem üzerinde nasıl uygulanabileceğini göstermiştir. Akademik iş birliklerin analizi için grafik veri yapıları etkin bir şekilde kullanılmış ve sonuçlar kullanıcı dostu bir arayüzle görselleştirilmiştir. Flask ve Cytoscape.js teknolojilerinin entegrasyonu, hem backend hem de frontend tarafında güçlü bir altyapı sağlamıştır

VI. YAZAR KATKILARI

Yusuf BÜLBÜL : Projenin backend tarafında çeşitli algoritmaların yazılması, geliştirilmesi ve uygulanmasında görev almıştır. Frontend koduna isterlerle alakalı güncellemeler ve iyileştirmeler yapmıştır. Projenin farklı durumlar için test edilmesi ve hataların düzeltilmesinde görev almıştır.

Saffet Hakan KOÇAK : Projenin frontend kısmında backendle uyumlu olacak şekilde görselleştirmeler ve iyileştirmelerde bulunmuştur. Projenin farklı durumlar için test edilmesi ve hataların düzeltilmesi sürecinde katkıda bulunmuştur. Projenin raporunun hazırlanmasında görev almıştır.

Her iki yazar proje boyunca iş birliği yapmış, süreci sabırlı ve istikrarlı bir şekilde yönetmişlerdir.

VII. KAYNAKÇA

https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/

https://medium.com/folksdev/binary-tree-veri-yap%C4%B1s%C4%B1nda-depth-first-search-mant%C4%B1%C4%9F%C4%B1-70482e00755c

https://youtube.com/playlist?list=PLY20HpFruiK17gQBHXjJN-Yi-FujiE1zVE&si=zWYX_gGGIiTy7mE9

https://youtube.com/playlist?list=PLfAfrKyDRWrG7tK01yW92A2j7Ou0qpOFm&si=fkZFuEOMK2SqjS4r

https://flask.palletsprojects.com/

https://pandas.pydata.org/

https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm

https://mathworld.wolfram.com/GraphTheory.html

https://flask-restful.readthedocs.io/

https://realpython.com/flask-by-example/

https://coderspace.io/sozluk/flask-nedir