Programlama
Lab.-1
3. Proje

Anıl Engin Keretli
Bilgisayar Mühendisliği Bölümü
Kocaeli Üniversitesi
Kocaeli, Türkiye
230201128

Ahmet Burak Karkaç
Bilgisayar Mühendisliği Bölümü
Kocaeli Üniversitesi
Kocaeli, Türkiye
220201173

I. ÖZET

Proje, akademik bir veri seti kullanarak yazarlar arasındaki işbirliği ilişkilerini bir graf yapısı üzerinden modellemeyi ve bu modellemenin sonucunda elde edilen analizlerle çeşitli veri yapılarını uygulamalı olarak sunmayı amaçlamaktadır. Çalışma kapsamında yazarlar düğüm, işbirlikleri ise kenarlarla temsil edilmektedir.

Proje, veri yapıları ve algoritma bilgisinin grafiksel analizle birleştirilerek görselleştirilmesini hedefler. Flask altyapısı ile geliştirilmiş olan backend uygulaması, çeşitli işlevler ve API uç noktaları sağlar. Kullanıcılara interaktif bir arayüz sunarak verilerin görsel ve dinamik bir şekilde analiz edilmesine olanak tanır.

II. GİRİŞ

Bilgi ve işbirliği analizleri, akademik çalışmalarda ve veri bilimi projelerinde yaygın bir şekilde kullanılır. Bu proje, bu kavramı uygulamak için akademik makale verilerini kullanan bir graf sistemi tasarlar. Projenin ana hedefleri şunlardır:

1. Yazarların ve işbirliklerinin bir graf modeli üzerinden analiz edilmesi.
2. Veri yapıları ve algoritmaların (Binary Search Tree, Kuyruk vb.) gerçek hayattan bir problemin çözümü için uygulanması.
3. Dinamik ve kullanıcı dostu bir grafik arayüzü geliştirilmesi.

III. YÖNTEM

Proje, çeşitli yazılım bileşenlerinden oluşan bir sistem üzerinde yürütülmüştür:

1. Veri İşleme

Veri ön işleme ve temizleme modülü, Excel dosyasından yüklenmiş yazar isimlerini standart hale getirmek ve analiz için temizlenmiş veri seti oluşturmak amacıyla geliştirilmiştir.

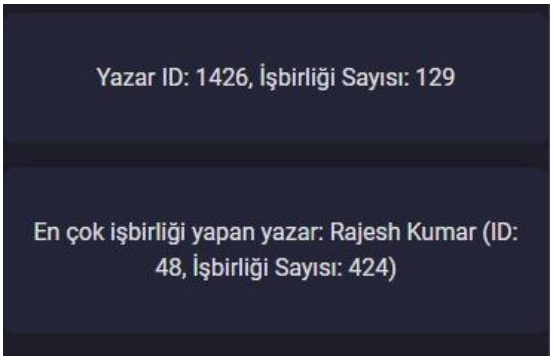
A. **Yazar İsimlerinin Standartlaştırılması:**
   - Yazar isimleri, fazla boşlukların ve noktalama işaretlerinin temizlenmesiyle birleştirilmiştir.
   - Benzer isimler, benzerlik oranı kullanılarak tespit edilmiş ve tekilleştirilmiştir.

B. **Co-authors Verisinin Parse Edilmesi:**
   - Co-authors kolonunda bulunan veriler string formatından listeye dönüştürülmüştür.
   - Tüm yazar isimleri temizlenmiş ve standartlaştırılmıştır.

C. **Benzerlik Kontrolü:**

- Yazar isimleri arasındaki benzerlik oranını hesaplamak için SequenceMatcher kullanılmıştır.
- İnitial ve tam isim uyumu kontrol edilmiş ve en uygun formata dönüştürülmüştür.

D. **Loglama:**
- Standartlaştırılan tüm yazar isimleri bir log dosyasında kayıt altına alınmıştır.



Kod Parçaları

- **Standartlaştırma:**

```
name = name.strip()
name = re.sub(r'[.,;:]', '', name)
name = re.sub(r'\s+', ' ', name)
```

- **Benzerlik Analizi:**

```
similarity = SequenceMatcher(None, first1, first2).ratio()
if similarity > 0.90:
```

- **Co-authors Parse:**

```
def parse_coauthors(self, coauthors_str: str) -> list:
    authors = coauthors_str.strip('[]').split(',')
    authors = [self.clean_author_name(author.strip().strip('"')) for author in authors]
    return [author for author in authors if author]
```

Elde Edilen Sonuçlar

- Toplamda **n** adet yazar ismi standart hale getirilmiş ve veri setinde tekrar eden isimler temizlenmiştir.
- Log dosyası oluşturularak tüm değişiklikler kaydedilmiştir.
- Yazar isimlerinin ısrarla karışması durumunda tam isim kullanılması tercih edilmiştir.

2. Graf Modelleme

Yazarlar düğüm, işbirlikleri ise kenarlar olarak modellenmiştir. Grafın temel özellikleri:

- **Düğüm Ağırlıkları:** Yazarların toplam makale sayıları.
- **Kenar Ağırlıkları:** Ortak makale sayıları.
- **Ortalama ve Standart Sapma:** Düğüm boyutlarının ve renklerin belirlenmesinde kullanılmıştır.

A. **Düğüm Sınıfı (Node):**
- Yazarın kimlik bilgileri, makaleleri ve bağlantıları saklanır.
- İşbirliği yapılan yazarların bağlantı ağırlıkları güncellenir.

B. **Graf Sınıfı (CollaborationGraph):**
- Yazar ve makale verileri graf yapısına dönüştürülür.
- Ortalama makale sayısı gibi metrikler hesaplanır.
- Dijkstra ve DFS algoritmaları ile en kısa ve en uzun yollar bulunur.

C. **Binary Search Tree (BST):**
- Yazarların makale sayısına göre sıralanmış bir ağaç oluşturulur.
- AVL algoritmaları ile dengeli bir yapı sağlanır.

Kod Parçaları

- **Dijkstra Algoritması:**

```
def dijkstra(self, start_id: int) -> tuple[dict, dict]:
    distances = {node_id: float('inf') for node_id in self.nodes}
    distances[start_id] = 0
    previous = {node_id: None for node_id in self.nodes}
    unvisited = set(self.nodes.keys())

    while unvisited:
        current = min(unvisited, key=lambda x: distances[x])
        if distances[current] == float('inf'):
            break
        unvisited.remove(current)

        for neighbor, weight in self.nodes[current].connections.items():
            if neighbor in unvisited:
                distance = distances[current] + (1.0 / weight)
                if distance < distances[neighbor]:
                    distances[neighbor] = distance
                    previous[neighbor] = current

    return distances, previous
```

- **BST Eklenmesi:**

```
def insert(self, key, data):
    if not self.root:
        self.root = BSTNode(key, data)
    else:
        self.root = self._insert(self.root, key, da
```

- **DFS ile En Uzun Yol:**

```
def find_longest_path(self, start_id: int) -> List[int]:
    visited = set()
    longest_path = []

    def dfs(current_id: int, current_path: List[int]):
        nonlocal longest_path
        visited.add(current_id)
        current_path.append(current_id)

        if len(current_path) > len(longest_path):
            longest_path = current_path.copy()

        for neighbor in self.nodes[current_id].connections:
            if neighbor not in visited:
                dfs(neighbor, current_path)

        current_path.pop()
        visited.remove(current_id)

    dfs(start_id, [])
    return longest_path
```

3. Algoritmalar

3.1 En Kısa Yol

Belirli iki yazar arasındaki en kısa yol, Dijkstra algoritması ile hesaplanmıştır.

3.2 Kuyruk ve BST Oluşturma

Yazarlar, düğüm ağırlıklarına göre sıralanan bir kuyruğa eklenir ve bu kuyruktan bir Binary Search Tree yapısı oluşturulur.

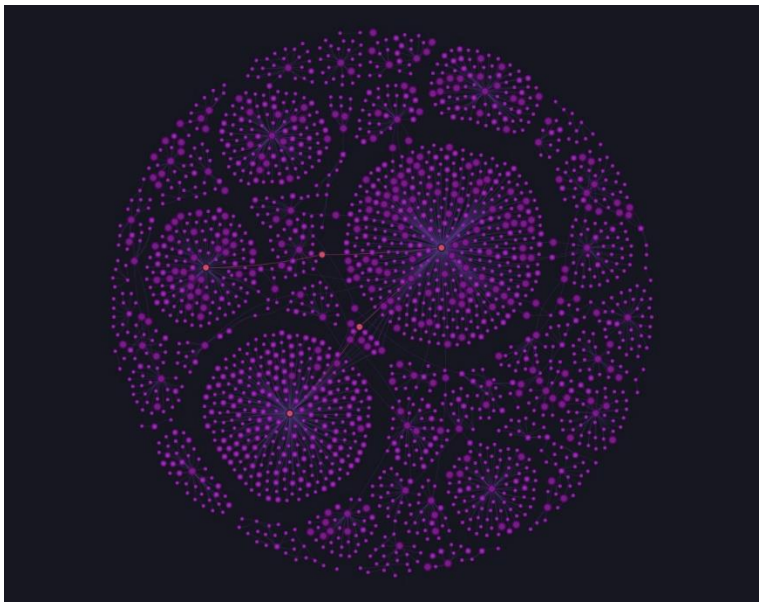
3.3 En Uzun Yol

Bir yazarın graf üzerinde gidebileceği en uzun yolun hesaplanması, DFS algoritması ile yapılmıştır.

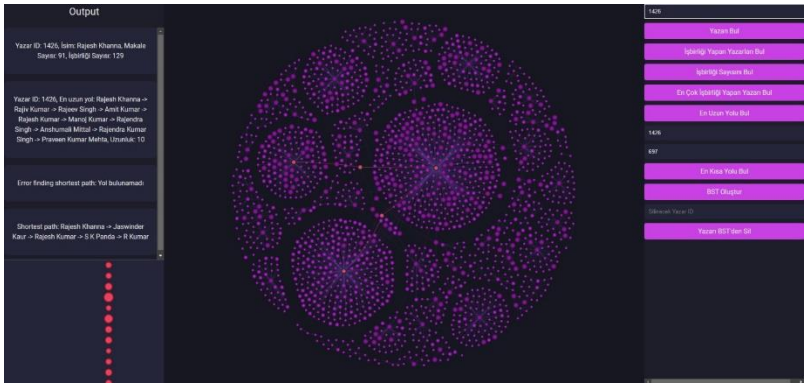
4. Düğüm ve Bağlantı Modeli

Projenin temel bileşenleri olan düğümler (Node) ve graf modeli (CollaborationGraph), akademik makalelerin yazarları ve işbirliklerini temsil etmek için özel olarak tasarlanmıştır:

- **Düğüm Yapısı:**
  - Her düğüm, bir yazarın kimliğini, adını, makalelerini ve diğer yazarlarla olan işbirliklerini tutar.
  - İşbirlikleri, ağırlıklı bağlantılar (kenarlar) ile temsil edilir.
- **Graf Yapısı:**
  - Makaleler ve yazarlar arasındaki ilişkiler graf yapısına eklenir.
  - Düğümler arası bağlantılar, ortak makalelere dayalı olarak oluşturulur.
  - Ortalama makale sayısına göre düğüm renkleri ve boyutları belirlenir.



IV. ARAYÜZ



React tabanlı bir kullanıcı arayüzü, verilerin dinamik olarak görselleştirilmesini ve etkileşimli analizler yapılmasını sağlar. Bu arayüz, Flask API'si ile entegre bir şekilde çalışır.

Uygulanan Yapılar

1. **Grafik Görselleştirme (GraphVisualization):**
   - Tüm yazarları ve işbirliklerini düğümler ve kenarlar ile gösterir.
   - En kısa yollar ve diğer öne çıkan yollar vurgulanabilir.

2. **Form ve Arama Alanları:**
   - Kullanıcılar, belirli bir yazarın ID'sini veya başlangıç ve bitiş noktalarını girerek analiz başlatabilir.
3. **Küçük Grafik Görselleştirme (SmallGraphVisualization):**
   - Özet grafiksel verileri, seçilen yazarların bağlantılarını ve diğer bilgileri görselleştirir.
4. **Veri Akışı Yönetimi:**
   - Kullanıcı girdileri, React state kullanılarak yönetilir.
   - Flask API'sinden gelen veri, state ile güncellenir ve bileşenlere aktarılır.

Kullanılan React Bileşenleri

- **GraphVisualization:** Tüm grafın görselleştirilmesi.
- **ShortestPathForm:** En kısa yol araması için form.
- **NodeInfo:** Belirli bir düğümün detaylı bilgilerini görüntüler.
- **MostCollaborativeAuthor:** En çok işbirliği yapan yazarı listeler.
- **CollaborationCount:** Belirli bir yazarın toplam işbirliği sayısını görüntüler.
- **LongestPath:** Belirli bir yazarın en uzun yolunu hesaplar ve gösterir.
- **AuthorCollaborators:** Bir yazar ve işbirlikçilerinin grafiğini oluşturur.
- **AuthorShortestPaths:** Bir yazarın işbirlikçileri arasında en kısa yolları görüntüler.
- **BSTVisualization:** İkili arama ağacının (BST) grafiksel temsili.

Kod Akışı

- **Veri Yükleme:**

```
useEffect(() => {
  const loadGraphData = async () => {
    try {
      const data = await fetchGraphData();
      setGraphData(data);
    } catch (err) {
      console.error('Error loading graph data:', err);
      setError('Error loading graph data');
    } finally {
      setLoading(false);
    }
  };
  loadGraphData();
}, []);
```

- **En Kısa Yol Arama:**

```
const handleShortestPathSearch = async () => {
  try {
    if (!startId || !endId) {
      setOutput(prev => [...prev, 'Start and end IDs are required']);
      return;
    }

    const response = await fetchShortestPath(startId, endId);
    if (!response || !response.path || !response.authors || !response.queue) {
      throw new Error('Invalid server response');
    }

    const { path, authors, queue } = response;
    setHighlightedPath(path);
    setOutput(prev => [...prev, `Shortest path: ${authors.join(' -> ')}`]);

    const finalNodes = queue.map(node => ({
      id: node.id,
      label: `${node.name}\n(${node.distance.toFixed(2)})`,
      color: '#e94560'
    }));

    setSmallGraphData({ nodes: finalNodes, edges: [] });
  } catch (err) {
    console.error('Error finding shortest path:', err);
    setOutput(prev => [...prev, `Error finding shortest path: ${err.message}`]);
  }
};
```

V. DENEYSEL SONUÇLAR

Proje, Flask tabanlı bir API ile desteklenmiştir. React arayüzü ile kullanıcılar, belirli yazarlar ve işbirlikçileri arasında analiz yapabilir:

1. **En Kısa Yol:** Kullanıcılar çift yazar ID'si girerek grafın en kısa yolunu görselleştirir.
2. **Düğüm Kuyruğu:** Düğüm ağırlıklarına göre kuyruk oluşturulur ve adımlar gösterilir.
3. **BST:** Kuyruktan oluşturulan Binary Search Tree, grafiksel bir formatta görselleştirilir.

VI. SONUÇ

Bu proje, yazarlar arasındaki işbirlikleri bir graf yapısında modelleyerek hem teorik hem de uygulamalı öğrenim sağlamıştır. Elde edilen React arayüzü ve Flask API'si, kullanıcıların akademik veri üzerinde etkili analizler yapmasını mümkün kılmıştır.

Projenin önerilen geliştirme alanları:

1. Daha büyük veri setlerinde performans optimizasyonu.
2. Kullanıcı dostu ve gelişmiş bir arayüz.
3. Gerçek zamanlı analizler için streaming verilerle entegrasyon.

REFERANSLAR

1. [1 A. Yazar ve B. Ortak, "Graf ve Algoritmalar: Temel Yaklaşımlar," İstanbul Yayınları, 2023.
2. Flask Dokumentasyonu, https://flask.palletsprojects.com/
3. VisJS Grafik Kütüphanesi, https://visjs.github.io/
4. React Dokumentasyonu, https://reactjs.org/