# Programlama Laboratuvarı 3.Proje Ödevi

1. Esra Kurt
*Bilgisayar Mühendisliği*
*Kocaeli Üniversitesi*
Kocaeli, Türkiye
esrakurt221@gmail.com

2. Zeynep Tandoğan
*Bilgisayar Mühendisliği*
*Kocaeli Üniversitesi*
Kocaeli, Türkiye
zeyneptandoğan03@gmail.com

***Özetçe*—Bu rapor, Kocaeli Üniversitesi Programlama Laboratuvarı 1 dersinin 1. proje ödevi için hazırlanmıştır.**

***Anahtar Kelimeler — Python,Veri Yapıları,Graf,İlgi Alanı***

## I. ÖZET

Bu proje; Python programlama dili kullanılarak veri yapılarını hazır kullanmadan ayrıca kendi oluşturduğumuz (gerçek olmayan) bir veri seti üzerinden dataları alarak Twitter kullanıcıları arasında belirli ilişkiler kurmayı ve bunları graf yapısı üzerinde göstererek belirli algoritmaları pekiştirmek amacıyla yapılmıştır.

## II. GİRİŞ

Birçok aşmadan oluşan bu proje Twitter verileri üzerinde çeşitli analizler yapmayı ve kullanıcılar arasında benzer ilgi alanlarına göre eşleşmeyi amaçlamaktadır. Öncelikle veri tabanı kullanılmadan Twitter api veya benzer uygulamaları kullanmamız veyahut dosyalarda tutulan verileri çekmemiz istenmektedir. Sonrasında kullanıcı nesnelerini tanımlamamız ve kullanıcı adı-soyadı, takipçi sayısı, takip edilen sayısı, dil, bölge, tweet içerikleri, takip edilenler ve takipçiler bilgilerini içermesi gerekmektedir. Sonrasın çekilen kullanıcı nesnelerine ait bilgileri hash tablosunda tutmamız gerektiği belirtilmiştir.

Kullanıcı ve takipçi, takip edilen ilişkilerini temsil eden bir graf oluşturmak amaçlanıp her kullanıcının birer düğüm(node), ilişkilerinin ise kenar(edge) ile gösterilmesi istenmiştir. Bununla birlikte kullanıcıların attığı tweetlere göre ilgi alanlarının kategorize edilerek hash tablosunda tutulmasıyla benzer ilgi alanlarına sahip kişilerin eşleştirilmesi gerekmektedir. Bu eşleştirmeler hash tabloları ve arama algoritmaları kullanılarak gerçekleştirilmelidir ve sonuçlar detaylarıyla metin dosyasına kaydedilmelidir. Sonrasında ise kullanıcılar arasındaki bağlantılar graf üzerinde analiz edilmeli ve BFS algoritması kullanılarak aynı seviyedeki kullanıcılar belirlenmelidir ve bunlar çıktıda verilmelidir.

Ek olarak belirli bir bölge ve dil için trend olan hastagler veya gündem olan konular listenlenmeli, tweet içeriklerindeki belirli anahtar kelimelere ve hashtaglar'e göre benzer tweetler DFS algoritmasına göre belirlenip listelenmelidir. Belirli iki kullanıcının takipçilerinden ilgi alnına göre filtreleme yapılarak ortak ilgi alnına sahip kullanıcılar belirlenip listelenmelidir. En son BFS ve Minimum Spanning Tree algoritmaları kullanılarak kullanıcılar arasında bağlantılar kurulup ilişkiler analiz edilmelidir.

## III. YÖNTEM

İlk olarak hangi dili kullanacağımıza ve geliştirme ortamını kullanacağımıza karar verdik. Python programlama dilinin bizim için daha verimli olacağını düşündüğümüzden onu kullandık. İlk projemizde Visual Studio Code kullandığımız için ve halihazırda bilgisayarlarımızda bulunduğu için tekrar kullanmayı kararlaştırdık. Ardından Twitter Api'yi araştırdık ve ücretsiz olarak sağladığı verinin yetersiz olduğunu gördük. Benzer uygulamalara bakındıktan ve verimsiz olacaklarını düşündüğümüzden dolayı yapay bir veri seti kullanmaya karar verdik ve proje grubunda paylaşılan on bin kullanıcılı, 170 bin tweetli veri setinden faydalandık. Ardından veri yapılarını araştırarak kodlamaya başladık.

1. Main class'ında ilk olarak gerekli modülleri ve dosyaları içe aktardık. "twitter_data.json" adlı dosyayı okutup içeriğini "twitter_data" değişkenine yükledik. "users, user_table, users_tweets, user_graph, language ve region gibi çeşitli veri yapılarımızı oluşturduk. Ana program(if__name__ == "__main__" : ) Twitter veri setindeki her kullanıcı için bir döngü başlatır. Her kullanıcı için Person nesnesi oluşturur ve bu nesneyi ve diğer bilgileri info fonksiyonuna gönderir. Bu her kullanıcı için bir info nesnesi oluşturur. Kullanıcının tweetlerini bir liste haline getirir ve bu listeyi users_tweets tablosuna ekler. Kullanıcıları bilgilerini user_table ve user_graph veri yapılarına ekler. Döngü tamalandığında, tüm kullanıcılar ve tweetler ilgili veri yapılarına yerleştirilmiş olur. Her kullanıcının ilgi alanlarını belirler ve bunları interests fonksiyonuna gönderir. İlk kullanıcının selamlaşmasını yazdırır ve same users fonksiyonunu çağırır. user_table'daki her kullanıcı için bir döngü başlatır ve kullanıcının dilini ve bölgesini language ve region tablolarına ekler. Bu sonuçları da "hastagler.json" dosyasına yazar. Kullanıcıdan bir isim alır ve bu ismi user_table'dan arar. Bu kullanıcının takipçilerini ve takip ettiklerini alır ve bu kullanıcı için bir grafik oluşturur. Takipçilerini ve takip ettiklerini grafiğe ekler ve bir

GraphVisualizer penceresi açar. Daha sonra users listesindeki
- her kullanıcı için bir döngü başlatır. Her kullanıcı için,
- kullanıcının takipçilerini ve takip ettiklerini alır. Her takipçi için, users_graph'a bir kenar ekler. Bu kenar, takipçiyi
- kullanıcıya bağlar. Her takip edilen kişi için, users_graph'a bir
- kenar ekler. Bu kenar, kullanıcıyı takip edilen kişiye bağlar.

Bu işlemler tamamlandığında users_graph tüm kullanıcıları ve aralarındaki ilişkileri içerir. Ardından BFS (Breadth-First Search) algoritmasını çağırır. Bu algoritma, users_graph'da belirli bir kullanıcıdan başlar ve grafiği genişlik öncelikli olarak gezerek aynı sayıda takipçiye sahip olan kullanıcıları bulur. Bu kullanıcıları same_followers tablosuna ekler. Son olarak bu tabloyu "bfs.json" dosyasına yazar.

Data stuctures class'ındaki Node sınıfı, bağlı listenin her bir düğümünü temsil eder. Her düğüm, bir anahtar (key), bir veri (data) ve bir sonraki düğüme işaret eden bir bağlantı (next) içerir. LinkedList sınıfı, bağlı listeyi temsil eder. İlk düğümü (head) saklar ve çeşitli işlemler gerçekleştirir: to_list: Bağlı listeyi bir Python listesine dönüştürür. insert: Yeni bir düğüm ekler veya var olan bir düğümün verisini günceller. find: Belirli bir anahtara sahip bir düğümü bulur. set: Belirli bir anahtara sahip bir düğümün verisini günceller. max: En yüksek veri değerine sahip düğümün anahtarını bulur. delete: Belirli bir anahtara sahip bir düğümü siler. create: Yeni bir düğüm oluşturur ve listenin sonuna ekler. kok sınıfı, bağlı listenin her bir düğümünü temsil eder. Her düğüm, bir veri (data) ve bir sonraki düğüme işaret eden bir bağlantı (next) içerir. myList sınıfı, bağlı listeyi temsil eder. İlk düğümü (head) saklar ve çeşitli işlemler gerçekleştirir: insert: Yeni bir düğüm ekler. __iter__: Listenin üzerinde yineleme yapmayı sağlar. find: Belirli bir veriye sahip bir düğümü bulur. extend: Mevcut listeyi başka bir listeyle genişletir. delete: Belirli bir veriye sahip bir düğümü siler. Queue sınıfı, bir kuyruk veri yapısını temsil eder. Ön (front) ve arka (rear) düğümleri saklar ve çeşitli işlemler gerçekleştirir: isEmpty: Kuyruğun boş olup olmadığını kontrol eder. enqueue: Kuyruğun sonuna yeni bir öğe ekler. dequeue: Kuyruğun başındaki öğeyi çıkarır. HashTable sınıfı, bir hash tablosu veri yapısını temsil eder. Bu yapı, anahtar-değer çiftlerini saklar ve hızlı veri erişimi sağlar. Bu sınıfın metotları şunları içerir: get_hash: Bir anahtarın hash değerini hesaplar. __getitem__: Belirli bir anahtara sahip bir değeri alır. get_items: Hash tablosundaki tüm öğeleri alır. find: Belirli bir anahtara sahip bir değeri bulur. __setitem__: Belirli bir anahtara sahip bir değeri ayarlar veya ekler. __delitem__: Belirli bir anahtara sahip bir değeri siler. create: Yeni bir anahtar oluşturur. find_common_words: En yaygın kelimeleri bulur. Binary_Tree sınıfı, bir ikili ağaç veri yapısını temsil eder. Bu yapı, düğümleri hiyerarşik bir şekilde saklar ve hızlı veri erişimi ve düzenlemesi sağlar. Bu sınıfın metotları şunları içerir: getRoot: Ağacın kök düğümünü alır. add: Ağaca yeni bir düğüm ekler. printTree: Ağacı belirli bir sıralamada yazdırır (in-order, pre-order veya post-order). Set sınıfı, bir küme veri yapısını temsil eder. Bu yapı, benzersiz öğeleri saklar ve hızlı veri erişimi ve düzenlemesi sağlar. Bu sınıfın metotları şunları içerir: add: Kümeye yeni bir öğe ekler. contains: Kümeye belirli bir öğenin olup olmadığını kontrol eder.

Proje_3 class'ımızda ilgi alanlarını sınıflandıran belirleyen hazır bir model kullandık. Öncelikle gerekli kütüphaneleri içe aktardık.Daha sonra modeli kendi kodumuza göre revize ederek yazdık. group_users_by_interest fonksiyonu, her kullanıcının tweetlerini analiz eder ve en ilgili olduğu konuyu belirler. Bu, her tweeti bizim belirlediğimiz kategorilerle(candidate_labels) karşılaştırarak ve en yüksek skora sahip olanı seçerek yapılır. Bu konuların her biri için, kaç kez çıktığını sayar ve en çok çıkan konuyu bulur. Bu fonksiyon daha sonra, her kullanıcıyı en çok ilgilendikleri konuya göre bir hash tablosuna (interests_users) ekler. Bu tablo, her konunun altında hangi kullanıcıların olduğunu gösterir. print_interests fonksiyonu, bu tabloyu alır ve "sonuclar.json" dosyasına yazar. Bu dosya, her konunun altında hangi kullanıcıların olduğunu gösterir.

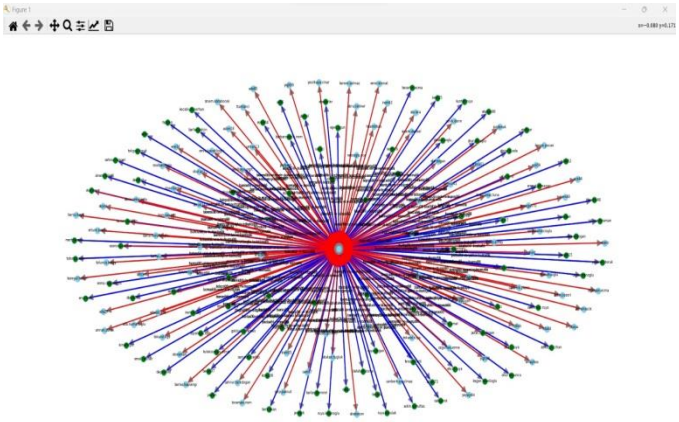
1. Proje_3 class'ımızda gerçekleşen en yüksek skoru bulma işlemi ise frekans class'ımızdaki işlemlere dayanır.Bu class'ta en_cok_gecen_kelimeler fonksiyonu, bir metin ve bir limit alır. Limit, döndürülecek en çok geçen kelimelerin sayısını belirler. Fonksiyon, metni küçük harflere çevirir ve boşluklara göre ayırarak bir kelime listesi oluşturur. Fonksiyon, belirli bir dizi yasaklı kelimeyi tanımlar. Bu kelimeler genellikle Türkçe'deki yaygın bağlaçlar, edatlar ve zarflardır.
2. Fonksiyon, bir HashTable oluşturur. Bu, her kelimenin kaç kez geçtiğini takip etmek için kullanılır. Fonksiyon, yasaklı kelimeleri filtreler ve kalan kelimeleri hash tablosuna ekler. Her kelime hash tablosunda bir anahtar olarak kullanılır ve değeri, kelimenin kaç kez geçtiğini gösterir. Son olarak, fonksiyon, hash tablosundaki en çok geçen kelimeleri bulur ve döndürür.

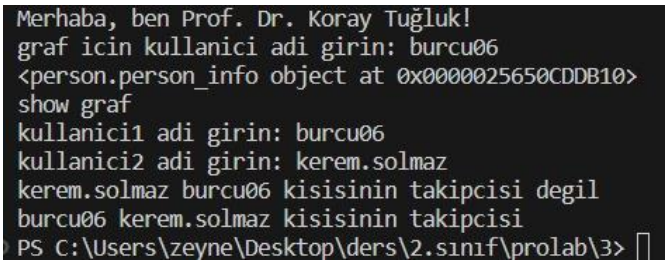
1. İlgi alanı class'ımızda belirli alanlara ait kelimeleri ve kullanıcıların ilgi alanlarını saklamak için veri yapıları oluşturduk. Node sınıfı, bağlı listenin her bir düğümünü temsil eder. Her düğüm, bir anahtar (key), bir değer (value) ve bir sonraki düğüme işaret eden bir bağlantı (next) içerir.
1. LinkedList sınıfı, bağlı listeyi temsil eder. İlk düğümü (head) saklar ve çeşitli işlemler gerçekleştirir: insert: Yeni bir düğüm ekler. find: Belirli bir anahtara sahip bir düğümü bulur. update: Belirli bir anahtara sahip bir düğümün değerini günceller. kelimeler adlı bir LinkedList oluşturulur ve her düğüm, belirli bir kategoriye ait kelimeleri içerir. Bu kelimeler, metin analizi için kullanılır. ilgi adlı bir HashTable oluşturulur. Bu tablo, her kullanıcının ilgi alanlarını saklar. Her kategori için ayrı bir HashTable oluşturulur (örneğin spor, sanat, cografya vb.) Bu tablolar, her kategorideki kullanıcıları saklar. interests fonksiyonu, bir kullanıcı ve bir metin alır. Bu metin, kullanıcının ilgi alanlarını belirlemek için analiz edilir. Her kelime, belirli bir kategoriye ait olup olmadığını belirlemek için kelimeler adlı bir LinkedList ile karşılaştırılır. Her kategori için, kaç kez çıktığını sayar ve en çok çıkan

kategoriyi bulur. Bu fonksiyon daha sonra, her kullanıcıyı en çok ilgilendikleri kategoriye göre bir hash tablosuna ekler. Bu tablo, her kategorinin altında hangi kullanıcıların olduğunu gösterir. Son olarak bu fonksiyon her kullanıcının gerçek adını, en çok ilgilendiği kategoriyi ve bu kategorinin kaç kez çıktığını içeren json dosyasına yazar. same_users fonksiyonu her kategorinin altında hangi kullanıcıların olduğunu gösteren dosyayı yazdırır.

Graf class'ımızda grafı görselleştirmek için PyQt5 penceresi oluşturduk. GraphVisualizer sınıfı, bir QWidget türevi olup, bir grafik, takipçiler ve takip edilenler listesi alır. Bu sınıf, bir grafik görselleştirme penceresi oluşturur ve çizer. initUI metodu, kullanıcı arayüzünü başlatır. Bir QVBoxLayout düzeni oluşturur, bir FigureCanvas oluşturur ve düzene ekler. drawGraph metodu, grafiği çizer. Öncelikle, bir networkx.DiGraph oluşturur ve grafiğin düğümlerini ve kenarlarını ekler.Ardından, networkx.spring_layout kullanarak düğümlerin konumlarını belirler. Takipçi ve takip edilenlerin kenarları belirlenir ve çizilir. Takipçi kenarları mavi, takip edilenlerin kenarları kırmızı renkte çizilir. Her düğüm için, eğer düğüm bir takipçi veya takip edilen ise, düğüm yeşil renkte çizilir ve düğümün adı etiket olarak eklenir. Son olarak FigureCanvas'ın düzeni sıkılaştırılır ve grafik gösterilir. LinkedListNode sınıfı bağlı listenin her bir düğümünü temsil eder. Her düğüm, bir anahtar (key) ve bir sonraki düğüme işaret eden bir bağlantı (next) içerir. Node sınıfı, graf düğümünü temsil eder. Her düğüm anahtar (key), bir değer (value) ve bağlantılı düğümler listesini (connections) içerir. Graph sınıfı graf veri yapısını temsil eder. Düğümler listesi (vertList) ve düğüm sayısı (numVertices) içerir. Bu sınıf, bir düğüm eklemek (addVertex), bir düğüm almak (getVertex), bir kenar eklemek (addEdge) ve tüm düğümlerin listesini almak (getVertices) için yöntemlere sahiptir. LinkedList_graph sınıfı bağlı liste veri yapısını temsil eder. Her liste bir baş düğüm (head) ve düğümlerin sözlüğünü (node_map) içerir. Bu sınıf, bir düğüm eklemek (insert), düğüm bulmak (find), tüm düğümlerin listesini almak (get_items) ve düğüm silmek (delete) için yöntemlere sahiptir. bfs fonksiyonu, genişlik öncelikli arama (BFS) algoritmasını uygular. Bu fonksiyon, bir başlangıç düğümü ve bir tablo alır. BFS algoritması kuyruk veri yapısını kullanarak çalışır. Başlangıç düğümü kuyruğa eklenir ve kuyruk boş olana kadar kuyruktan bir düğüm çıkarılır ve ziyaret edilir. Daha sonra ziyaret edilen düğümün tüm komşuları ziyaret edilmedilerse kuyruğa eklenir. Bu işlem, kuyruk boş olana kadar devam eder. Bu kod parçacığı BFS algoritmasını uygular ve aynı takipçi sayısına sahip kullanıcıları gruplandırır.

## IV. DENEYSEL SONUÇLAR



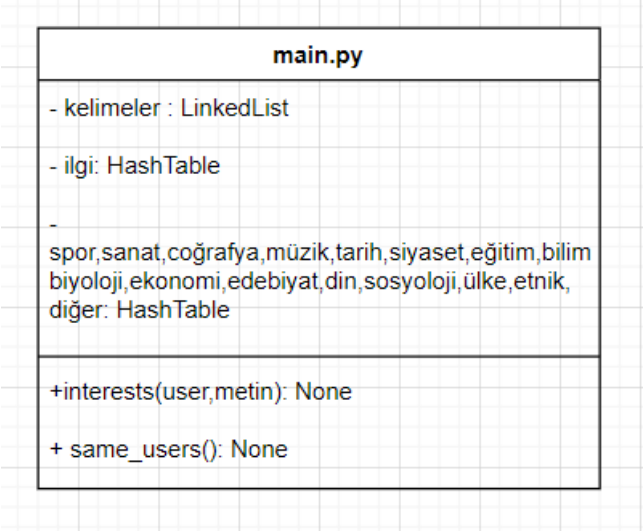

## SONUÇ

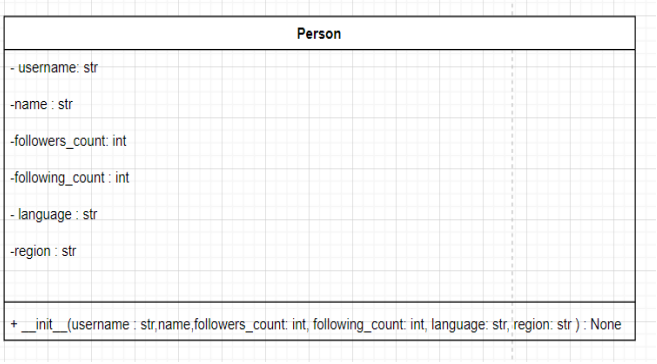
Sonuç olarak; bu projede çeşitli veri yapıları kullanarak kullanıcıların tweetlerini analiz edip ilgi alanlarına göre gruplandırdık ve sonuçları json dosyasına yazdırdık. Ayrıca kullanıcılar arasındaki bağlantıları görselleştirerek grafik arayüzü kullandık.

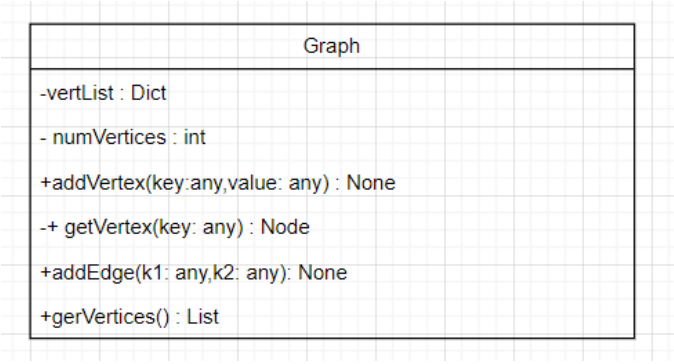
## YAZAR KATKILARI

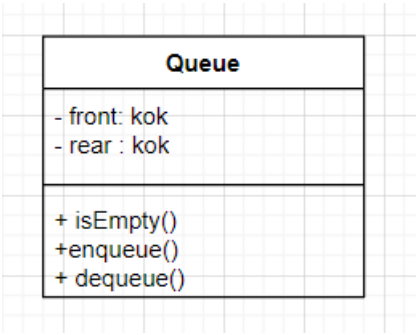
Projenin her aşamasında beraber olduğumuz için birlikte yapıldı. Veri çekme araştırmaları ve rapor detaylarına Esra Kurt ağırlık verirken Veri Yapıları araştırmaları, Graf oluşturma kısmına Zeynep Tandoğan yoğunlaştı.

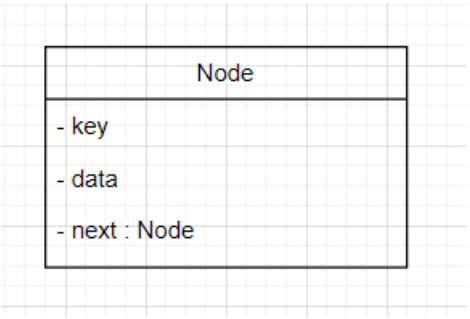
## UML DİYAGRAMLARI

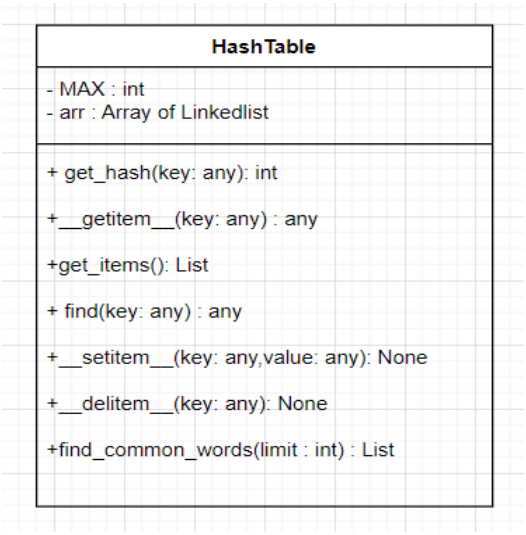


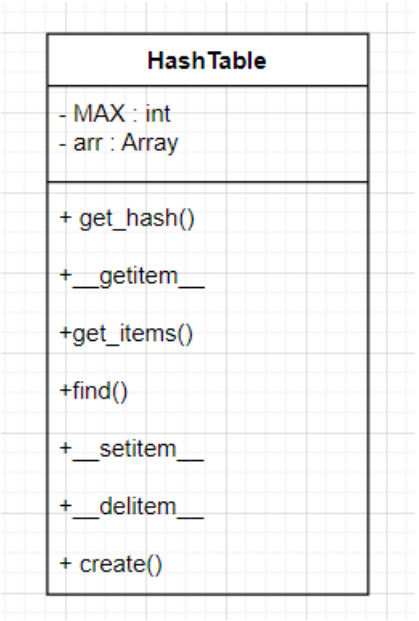



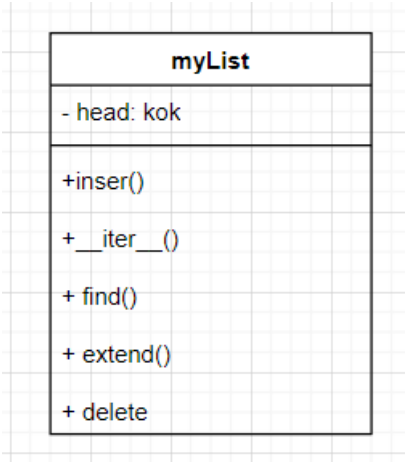



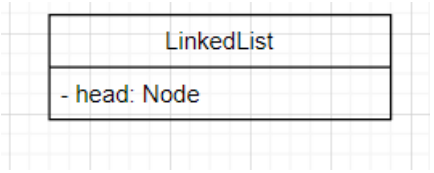

Queue
- front: kok
- rear : kok
+ isEmpty()
+enqueue()
+ dequeue()


Node
- key
- data
- next : Node


HashTable
- MAX : int
- arr : Array of Linkedlist
+ get_hash(key: any): int
+__getitem__(key: any) : any
+get_items(): List
+ find(key: any) : any
+__setitem__(key: any,value: any): None
+__delitem__(key: any): None
+find_common_words(limit : int) : List

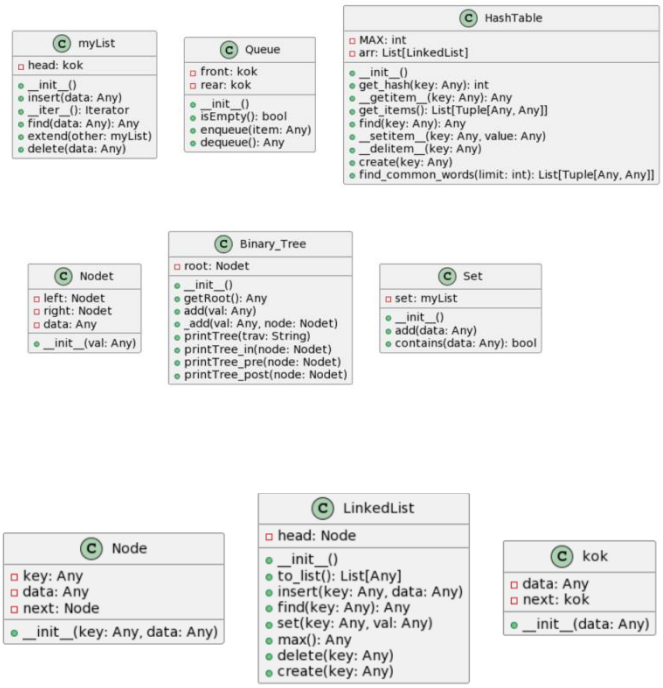

HashTable
- MAX : int
- arr : Array
+ get_hash()
+__getitem__
+get_items()
+find()
+__setitem__
+__delitem__
+ create()

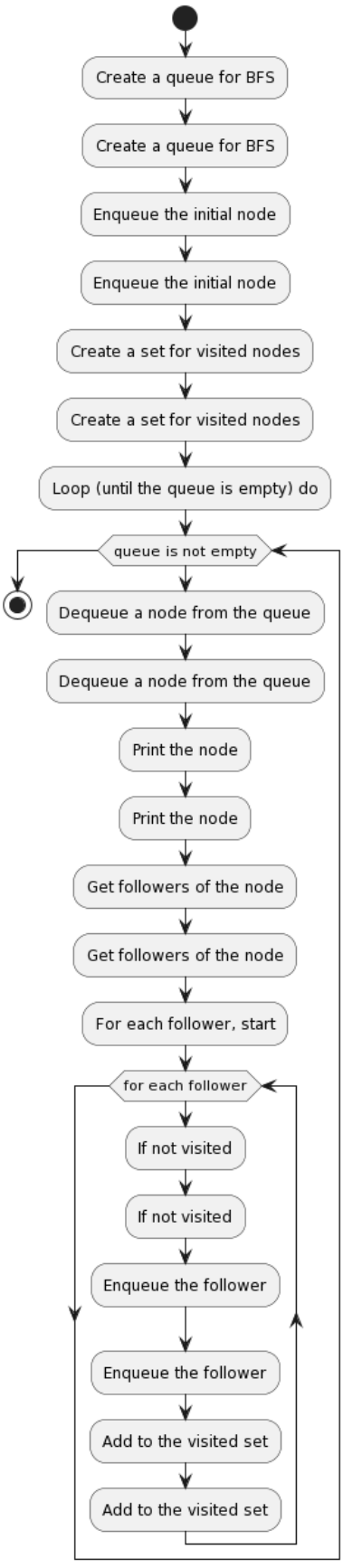

myList
- head: kok
+inser()
+__iter__()
+ find()
+ extend()
+ delete

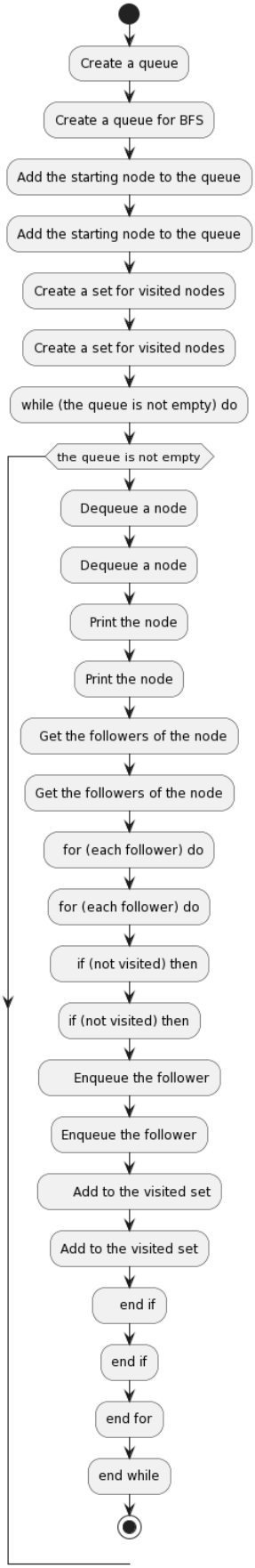

LinkedList
- head: Node

AKIŞ DİYAGRAMLARI

Bu sayfaya sığmadığı için aşağıda verilmiştir.




Create a queue for BFS
Create a queue for BFS
Enqueue the initial node
Enqueue the initial node
Create a set for visited nodes
Create a set for visited nodes
Loop (until the queue is empty) do
queue is not empty
Dequeue a node from the queue
Dequeue a node from the queue
Print the node
Print the node
Get followers of the node
Get followers of the node
For each follower, start
for each follower
If not visited
If not visited
Enqueue the follower
Enqueue the follower
Add to the visited set
Add to the visited set


Create a queue
Create a queue for BFS
Add the starting node to the queue
Add the starting node to the queue
Create a set for visited nodes
Create a set for visited nodes
while (the queue is not empty) do
the queue is not empty
Dequeue a node
Dequeue a node
Print the node
Print the node
Get the followers of the node
Get the followers of the node
for (each follower) do
for (each follower) do
if (not visited) then
if (not visited) then
Enqueue the follower
Enqueue the follower
Add to the visited set
Add to the visited set
end if
end if
end for
end while

Kaynaklar


[1] https://stackoverflow.com/questions/8624779/my-python-program-is-very-slow-how-can-i-speed-it-up-am-i-doing-something-wron

[2] https://github.com/anantkaushik/Data-Structures-and-Algorithms

[3] https://github.com/tidwall/hashmap.c

[4] https://granulate.io/blog/optimizing-python-why-python-is-slow-optimization-methods/

[5] https://www.google.com/search?q=jasj+function+python&oq=jasj+function+python&gs_lcrp=EgZjaHJvbWUyBggAEEUYOTIICAEQABgWGB4yCAgCEAAYFhgeMggIAxAAGBYYHjIICAQQABgWGB4yCAgFEAAYFhgeMggIBhAAGBYYHjIICAcQABgWGB4yCAgIEAAYFhgeMgoICRAAGA8YFhgeMggIChAAGBYYHtIBCDY1ODJqMGo3qAIAsAIA&client=ms-android-samsung-ga-rev1&sourceid=chrome-mobile&ie=UTF-8

[6] https://github.com/anantkaushik/Data-Structures-and-Algorithms

[7] https://medium.com/@ibrahimirdem/pythonda-json-veri-okuma-ve-olu%C5%9Fturma-ade3e33f6184

[8] https://ceaksan.com/tr/python-tweepy-twitter-api

[9] 7. https://olympus.mygreatlearning.com/courses/50631