Yazılım Laboratuvarı Web İndeksleme Uygulaması

Özge POYRAZ
Kocaeli Üniversitesi
Mühendislik Fakültesi
Bilgisayar Mühendisliği
180202025@kocaeli.edu.tr

Burak Can TEMİZEL
Kocaeli Üniversitesi
Mühendislik Fakültesi
Bilgisayar Mühendisliği
180202024@kocaeli.edu.tr

***Özet*—Bu çalışmada, internet siteleri üzerinde çeşitli hesaplamalar ile indeksleme işlemi gerçekleştiren bir web uygulaması geliştirdik. Uygulama kelimelerin frekanslarını hesaplama, frekanslar üzerinden anahtar kelimeleri bulma, benzerlik hesaplama, elde edilen benzerlikleri ve anahtar kelimeleri kullanarak indeksleme yapma, yine bu veriler ve makine öğrenmesi modeli ile semantik analiz gerçekleştirme gibi çeşitli fonksiyonlara sahip. Uygulamayı geliştirirken Python programlama dili ve Flask frameworkünü kullandık.**

Anahtar kelimeler—Web, indeks, frekans, benzerlik, anahtar kelime, semantik analiz, doğal dil işleme, Python, Flask.

I. GIRIŞ VE PROBLEM TANIMI

Gerçekleştirdiğimiz projenin tanımını yapmamız gerekirse kullanıcı tarafından bir web sitesi üzerinden girilen web site ve web site kümeleri arasında çeşitli analiz yöntemleri ile frekans hesaplama, anahtar kelime bulma, benzerlik bulma, indeksleme ve semantik analiz işlemleri bir sunucuda gerçekleştiriliyor. Daha sonrasında sunucudan elde edilen veriler yine aynı web sitesi üzerinden kullanıcıya sunuluyor.

Web sitesi kısmında Html, tasarım için Pure.css , ağaç görselleştirmesi için Javascript ve Json Tree-view kütüphanesi, Flasktan yolladığımız verileri serve etmek için Jinja Template Engine, sunucu tarafında ise Flask web server kullanıldı.

Uygulama içerisinde dil ve dil özelliklerine bağlı birtakım işlemler gerçekleştirdiğimiz için, kararlı bir yapı elde etmek amacıyla hedef dilimizi İngilizce olarak belirledik. Bu şekilde dilin kurallarını daha net bir şekilde uygulayıp daha etkili sonuçlar alabileceğimizi düşündük.

II. YÖNTEMLER, YAKLAŞIMLAR VE PROGRAM MIMARISI

Projeyi temel olarak yukarıda bahsettiğimiz beş kısıma ayırabiliriz. Bu kısımda programın farklı özelliklerini oluşturmak için kullandığımız araçlar ve yöntemler üzerinde durularak ayrıntılı olarak bilgi verilecektir. Program mimarisi daha detaylı bir şekilde açıklanacaktır..

A. Frekans Hesaplama

Bu aşamada hedefimiz kullanıcıdan aldığımız url adresinde geçen her kelimenin kaç defa tekrar edildiğini hesaplamaktı. İlk olarak Flask ile web serverı gerçekledik. Daha sonrasında static serve yöntemi ile sunucu üzerinden kullanıcının etkileşime girebileceği, statik bir html sayfası tasarladık. Tasarımı gerçeklerken Pure.css kullandık. İlk aşama için form içerisinde bir adet textbox üzerinden url bilgisini aldık ve buton aracılığı ile sunucuya post ettik. Sunucu tarafında post edilen url verisine bir get requesti göndererek html contentini aldık. Daha sonrasında Beautifulsoup kütüphanesi ve lxml parser aracılığı ile site içeriğini düz yazı olarak biçimlendirdik. Düz yazı haline strip ve lower fonksiyonlarını uygulayarak, daha sonrasındaki işlemlerimiz için standart hale getirdik. Elde ettiğimiz bu yazı verisini nltk kütüphanesinin tokenizerını kullanarak, regex ifadesi yardımıyla kelimelere ayırdık. Bu noktadan itibaren elimizde düzensiz bazı kelimeler bulunmaktaydı. Frekans hesaplama işlemine geçmeden önce bu fonksiyona özel bir filtreme metoduyla kelimeleri filtreden geçirdik. Bir harften daha kısa kelimeler yok sayıldı. İngilizcede "a" ifadesi tek harfli anlamlı özel bir ifade olduğu için hariç tutuldu. Yine benzer şekilde İngilizce dilinde en uzun kelime 44 karakter olduğu için 44 karakterden uzun kelimeler filtrelendi. Rakam ve sayılar da yok sayıldı.

Artık elimizde filtre edilmiş kelimeler bulunmaktaydı. İsterin asıl amacını gerçekleştirmek için bu kelimeleri sayan bir metot geliştirdik. Bu metot bir dict objesi içerisine kelimeleri eğer mevcut değiller ise ekliyor, mevcutlar ise adetlerini arttırıyordu. Bu şekilde frekansları hesaplamış olduk. Daha sonrasında bu dicti sorted metotu yardımı ile sıralayarak, frekansları büyükten küçüğe azalan bir hale getirdik. Son olarak sunucuya gelen post requestine karşılık olarak elde ettiğimiz bu listeyi web sitesine gönderdik. Jinja Template Engine ifadeleri ile html içerisinde tablo halinde yazdırılmasını sağlayarak, frekans hesaplama işlemini gerçekleştirmiş olduk.

Word	Count
the	1790
of	1012
and	767
in	685
turkey	530
a	299
to	285

Frekans Tablosundan Kesit

B. Anahtar Kelime Bulma

Bu aşamada hedefimiz kullanıcıdan aldığımız url adresindeki anahtar kelimeleri tespit etmekti. Yapı olarak frekans hesaplama ile benzer olmasına karşın bazı ek özellikler içermekteydi. Öncelikle ilgili statik html sayfası hazırlandı. Bu aşamada Url alıp filtreleme kısmına kadar frekans hesaplama ile aynı şekilde ilerlemekte fakat anahtar kelime bulurken ek olarak ayrı bir filtreleme işlemi daha gerçekleştirilmektedir. Stopwords.py içerisinde İngilizce dilinde bulunan ve çok sık tekrar edilen bazı ifadeler liste halinde mevcuttur. Bu ifadelerin varlığı benzerlik hesaplama gibi işlemlerde hatalara yol açacağı için kaldırılmaları gereklidir. Ayrı bir metot yardımı ile frekans için kullandığımız kelime dizisi tekrardan bir filtrelemeden

geçirilerek bu kelimelerden arındırılmaktadır. Azalan frekans sırasına göre dizilen bu listeden seçilen on adet en yüksek frekanslı kelime bizim anahtar kelime setimizi oluşturmaktadır.

Word	Count
turkey	530
retrieved	269
turkish	265
ottoman	104
world	86
original	85
archived	84
european	70
august	68
february	68

Anahtar Kelime Tablosundan Kesit

C. *Benzerlik Hesaplama*

Bu aşamada farklı olarak kullanıcıdan bir url ile birlikte bir de url kümesi alıyoruz. Daha sonrasında bu urlnin küme içerisinde bulunan her url ile olan benzerliğini hesaplamamız gerekiyor. Web sayfası üzerindeki bir form aracılığı ile ilgili veriler sunucuya post ediliyor. Daha sonrasında sunucuda url ve url kümesi ayıklanıyor. Url kümesindeki her url bir listeye ekleniyor. Daha önceden hazırladığımız anahtar kelime bulma fonksiyonu ile ana url için anahtar kelime tespiti gerçekleştiriliyor. Daha sonrasında hazırladığımız bir fonksiyona ana url, ana url'nin anahtar kelimeleri ve url kümesi yollanıyor. Bu metot bize küme elemanlarının ana urlmiz ile olan benzerliklerinin olduğu veri yapısını ve küme içerisindeki her eleman için anahtar kelime listelerinin olduğu veri yapısını dönecektir. Bu fonksiyon içerisinde kendisine gelen url kümesindeki her eleman için anahtar kelimeleri bulur. Kendisine gönderilmiş ana url anahtar kelimeleri ile bu anahtar kelimeler arasında bir benzerlik formülü yardımı ile benzerlik skorunu hesaplar ve döndürür. Benzerlik hesaplanırken ortak olan anahtar kelimelerin frekansları her iki url içerisinde ayrı ayrı kareleri alınarak toplanır. Daha sonrasında genel toplamların karekökü alınır. Daha sonrasında bu iki değer birbiriyle çarpılır ve karelerinin toplamından çarpımlarının farkı alınarak bu değere bölünür. Elde ettiğimiz değer bize normalize bir değerdir. Bir sabit ile çarpılarak istenilen aralığa genişletilir. Aşağıda matematiksel ifade daha net bir şekilde verilmiştir.

$$a = \{\text{Ortak Anahtar Kelimelerin Birinci Sitedeki Frekansları}\}$$

$$b = \{\text{Ortak Anahtar Kelimelerin İkinci Sitedeki Frekansları}\}$$

$$aa = \sqrt{\sum_{i}^{n} a[i]^2}$$

$$bb = \sqrt{\sum_{i}^{n} b[i]^2}$$

$$benzerlik = \frac{aa * bb}{aa^2 + bb^2 - aa * bb} * 100$$

Benzerlik Formülü

Bu fonksiyon sonucu elde edilen veriler web sayfasına post edilir. Burada anahtar kelimeler ve benzerlikler tablolaştırılarak kullanıcıya sunulur.

Main Site Keywords

Word	Count
ottoman	613
empire	349
turkish	131
history	106
century	86
state	80
war	79
sultan	75
press	68
university	66

Similarity Table

Site	Including Keywords		Similarity Score
https://en.wikipedia.org/wiki/Turkey	turkey	530	60.383416162393274
	retrieved	269	
	turkish	265	
	ottoman	104	
	world	86	
	original	85	
	archived	84	
	european	70	
	august	68	
	february	68	

Benzerlik Sayfasından Kesit

D. *İndeksleme*

Bu aşamada daha önce elde ettiğimiz tüm verileri kullanarak yine bir web sitesi kümesine bir urlnin benzerlik oranını ölçecektik fakat farklı olarak url kümesi içerisindeki alt urlleri de indeksleyerek onların da benzerliğe olan katkısını bulmamız gerekiyordu. Yine aynı şekilde bu fonksiyona özel bir web sayfası tasarlayarak başladık. Daha sonrasında bir form üzerinden ilgili verileri web sunucumuza post ettik. Bir url ile birlikte bir de url kümesini sunucuya aktarmış olduk. Ana url için anahtar kelimeleri çıkarttık. Daha sonrasında bir ağaç yapısı için kök düğümü oluşturduk. Bir fonksiyon yardımıyla recursive olarak ana url adresinden başlayıp belli bir seviye kadar dolaşıp alt urllere erişip gerekli işlemleri yapacaktık. Recursive fonksiyonumuz 7 adet parametre alıyordu. Ana url adresi , ana url için önceden hesaplanmış anahtar kelime listesi, üst düğüm, diğer url listesi, seviye, dallanma adeti, ağaç kökü. Burada üst düğüm ve ağaç kökü referansları ağaca ekleme işlemleri için gerekirken, seviye ve dallanma adeti recursive fonksiyonun ilerlemesi ve durması için kullanılan değerlerdir.

Dokümanda belirtildiği üzere fonksiyon birinci seviyeden başlayıp üçüncü seviyede durur. Normalde sayfa içerisindeki bütün urlleri çıkartması gerekirken karşılaştığımız time complexity problemi yüzünden dallanmaya sınır koymamız gerekti. Burası ile ilgili ayrıntılı bilgi eksiklikler kısmında verilecektir. Fonksiyonun çalışma mantığına daha yakından bakmamız gerekirse öncelikle diğer url listesindeki tüm elemanlara her dallanmanın başında toplu bir halde request atarak yarı-asenkron çalışır. Bu şekilde sync olarak request atılmadığı için bekleme süresinden büyük bir kar edilir. Eğer uygun seviyenin altındaysak benzerlik fonksiyonu sayesinde daha benzerlik hesaplama kısmında olduğu gibi tüm url kümesi için ana urlye olan benzerlikler hesaplanır ve hepsi için anahtar kelime listeleri çıkartılır. Fonksiyon tekrar kendini çağırırken requestten elde ettiği response'u bir alt link bulma fonksiyonuna göndererek dallanmayı gerçekleştirir. Ayrıca seviyesini de arttırır. Bu alt link bulma fonksiyonu parametre

olarak aldığı request nesnesindeki urlleri çıkartarak bir liste haline getirir. Uygun durma noktasında kendini çağrım sonlanır.

Bu noktaya kadar sadece ana url ile küme arasındaki tüm elemanlar için olan benzerlik hesaplanmıştır. Bizim için önemli olan alt daldaki benzerlik oranlarının üst dallara katkı sağlaması. Bu yüzden ağaç yapısından bir dict yapısı oluşturuldu. Daha sonrasında bir metot yardımıyla ağaç dolaşılarak hesaplanan skorların lokal skorlar olarak varsayılması ve her lokal skorun üst dallarına toplam olarak etki etmesi sağlandı. Daha sonrasında ağaç yapısı json haline getirildi. Son olarakta site sıralamalarını gösterebilmek adına sıralı bir dict haline getirildi. Bütün elde edilen veri yapıları sunucu tarafından web sayfasına postlandı ve orada kullanıcıya sunuldu. Tabloların sunumunda jinja kullanılırken ağaç yapsısını sunabilmek için json tree-view adında harici bir kütüphane kullanıldı. Bu şekilde indeksleme aşamasını tamamlamış olduk.


```
{
  name: "root",
  children: [
    {
      name: {
        url: "https://en.wikipedia.org/wiki/Ottoman_Empire",
        localScore: 60.383416162393274,
        generalScore: 762.1179142596054,
        keywords: {
          ottoman: 613,
          empire: 349,
          turkish: 131,
          history: 106,
          century: 86,
          state: 80,
          war: 79,
          sultan: 75,
          press: 68,
          university: 66
        }
      },
      children: [
        {
          name: {
            url: "https://en.wikipedia.org/wiki/Ottoman_Caliphate",
            localScore: 32.44997806103254,
            generalScore: 226.92999894085807,
            keywords: {
              ottoman: 73,
              empire: 42,
              caliphate: 34,
              ii: 20,
              movement: 18,
              edit: 17,
              sultan: 16,
              turkish: 15,
              mehmed: 15,
              islam: 13
            }
          },
          children: [
            {
              name: {
                url: "https://en.wikipedia.org/wiki/Rashidun_Caliphate",
                localScore: 0,
                generalScore: 0,
                keywords: {
                  al: 105,
                  umar: 83,
                  abu: 76,
                  caliphate: 71,
```


Site	Similarity Score(General)
https://en.wikipedia.org/wiki/Ottoman_Empire	762.1179142596054
https://en.wikipedia.org/wiki/Ottoman_Caliphate	226.92999894085807
https://en.wikipedia.org/wiki/Ottoman_Empire#mw-head	172.0684622848119
https://en.wikipedia.org/wiki/Ottoman_Empire#searchInput	172.0684622848119
https://en.wikipedia.org/wiki/List_of_Turkic_dynasties_and_countries	69.68768855213149
https://en.wikipedia.org/wiki/Caliphate	69.19664859536717
https://en.wikipedia.org/wiki/File:Wikipedia_-_Ottoman_Empire_-_Main_History.ogg	60.979886034598785

İndeks Ağacı ve Benzerlik Tablosundan Kesit

E. Semantik Analiz

Bu aşamada anahtar kelimeler üzerinde semantik analiz gerçekleştirip alakalı kelimeleri bulmamız gerekiyordu. Başta belirttiğimiz gibi hedef dil olarak İngilizce'yi belirlediğimiz için bu dil üzerine kurulmuş doğal dil işleme modellerini kullanarak anlamlı kelimeleri bulabilirdik. Bu noktada doğal dil işleme ve makine öğrenmesi modellerine bakınmayı başladık. Nltk kütüphanesi içindeki wordnet modelini anlamlı kelimeleri bulmak için kullanmaya karar verdik. İlk olarak web sayfasını tasarladık daha sonrasında diğer aşamalarda olduğu gibi gerekli girdileri sunucuya post ettik. Ana urlmiz için anahtar kelimeleri bulduk. Daha sonrasında anlamlı kelimeleri bulmak için bir fonksiyon yazdık. Bu fonksiyon normal anahtar kelime listesini alıyor ve her kelime için maksimum iki tane olmak üzere anlamlı kelime buluyor. Burada önemli nokta benzerlik hesaplarken frekans değerlerinden faydalandığımız için bulunan anlamlı kelimeye, anahtar kelimelerdeki frekans karşılığı veriliyor. Örneğin siyah kelimesi 50 frekans değeriyle işlem görecekse, anlamlı kelime olarak bulunan kara kelimesi içerikte geçmese dahi siyah ile anlamca yakın olduğu için bu 50 frekans değerine sahip oluyor. Bu şekilde benzerlik kısmında daha yüksek tutarlılıkta sonuç alabileceğimizi düşündük. Bu noktadan sonra elimizde anahtar kelimeler ve benzer kelimelerin olduğu birleşmiş bir dict yapısı oluşuyor. Bizden istenen ise bu yapıyı kullanarak indeksleme kısmında gerçekleştirdiğimiz benzerlik bulma işlemini yinelemek. Aynı recursive yapıyı fonksiyonları anlamlı anahtar kelime listelerini diğer urller için de çıkaracak şekilde düzenleyip tüm işlemleri benzer şekillerde gerçekleştirerek her site için benzerlik skorunu elde ediyoruz. Daha sonrasında elde ettiğimiz verileri web sayfasına postlayarak aynı şekilde görselleştirerek kullanıcıya sunuyoruz.

Word	Count	Semantic 1	Semantic 2
soviet	577	soviet	None
union	247	union	labor_union
war	198	war	warfare
russian	145	russian	None
ussr	128	soviet_union	russia
russia	116	soviet_union	russia
world	110	universe	existence
stalin	104	stalin	joseph_stalin
archived	103	archive	file_away
retrieved	103	recover	retrieve

Site	Similarity Score(General)
https://en.wikipedia.org/wiki/Nazi_Germany	573.8437386789035
https://en.wikipedia.org/wiki/Nazi_Germany#searchInput	191.28124622630116
https://en.wikipedia.org/wiki/Nazi_Germany#mw-head	191.28124622630116
https://en.wikipedia.org/wiki/Das_Dritte_Reich	95.64062311315058
https://en.wikipedia.org/wiki/Wikipedia:Protection_policy#extended	0
https://en.wikipedia.org/wiki/Wikipedia:Good_articles	0

Anlamlı Kelimeler ve Benzerlik Tablosundan Kesit

III. KAZANIMLAR

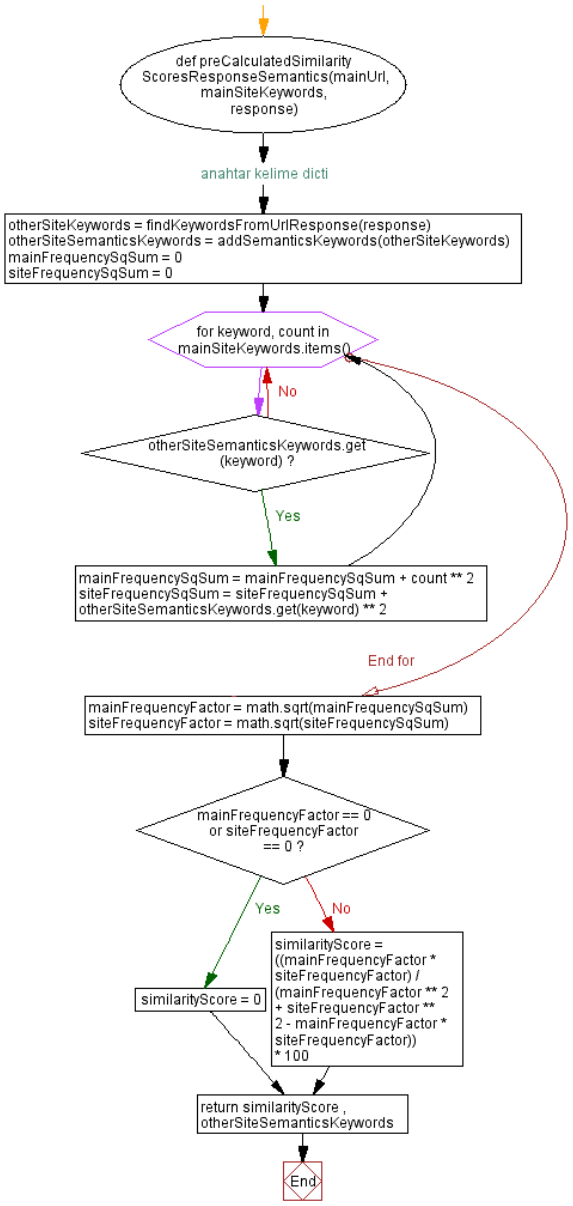
Bu projeyle, ilk defa bir web uygulaması gerçekleştirdik. Web teknolojileri hakkında bilgiler edindik. Farklı araçlar tanıdık ve kullandık. Doğal dil işleme kütüphanelerinin genel işleyici hakkında araştırmalar yapıp temel çalışma mantığını anladık. Kendi sunucumuzu yazıp wen sitemiz ile haberleşmesini sağladık. Çeşitli web indeksleme işlemleri gerçekleştirdik.

IV. EKSIKLIKLER

Projede dördüncü adımda recursive olarak tüm alt urlleri bulurken bir time complexity problem ile karşılaştık çünkü programımız sync olarak ilerliyordu ve her bir işlem için atılan bir requestin cevabını beklemesi gerekiyordu. Bu da tüm urlleri gezinmeyi imkansız bir hale getiriyordu. Daha sonrasında her dallanmada tüm requestleri bir araya toplayıp hepsini toplu halde atarak çok büyük ölçüde performans artışı elde ettik. Bundan önce beş dallanma limitiyle dakikalar süren indeksleme işlemini daha büyük dallanma değerleriyle çok daha kısa değerlere indirdik. İşlemi saniyeler civarına indirebilmek için tüm program asenkron bir hale getirmemiz gerektiğini fark ettik. Fakat her ne kadar

daha önce asenkron programlama gerçekleştirdiysekte projeyi daha kompleks bir hale taşımamak için şu an olduğu yarı-asenkron halde bırakmaya karar verdik. Sınırlı bir dallanma limitiyle yine çok uzun olmayan süreler içerisinde istediğimiz indeksleme işlemlerini gerçekleştirmeyi başardık. Projenin tasarımı daha fazla iyileştirilebilirdi ama deneysel bir proje olduğu için son Kullanıcı deneyimini çok iyileştiremedik ve pure.css gibi minimalist bir kütüphane ile sade web sayfaları tasarladık.

V. Akış Diyagramları ve Yalancı Kod



Benzerlik Formülünün Akış Şeması

VI. Geliştirme Ortamı ve Kullanılan Dil

Projeyi geliştirirken Python programlama dili ve Flask Frameworkünü kullandık. Projeyi Windows İşletim Sistemi üzerinde gerçekleştirirken, geliştirme ortamı olarak Visual Studio Code Editorunu kullandık.

VII. Deneysel Sonuçlar

Web uygulaması geliştirirken yaşanan test aşamalarının masaüstü uygulaması geliştirmekten farklı yanları bulunuyordu. Uygulamayı test ederken sürekli olarak web server kontrolü gerçekleştirdik. Daha önce bahsettiğimiz time complexity sonucu oluşan performans problemini çözerken çeşitli profilleme araçları ile çeşitli ölçümler gerçekleştirerek bu konuda oldukça pratik elde ettik. Projede zamanı minimize etmek için fonksiyonlar içinde çok sayıda analiz gerçekleştirip, çözümlemeye çalıştık. Doğal dil işleme hakkında teorik araştırmalar yapıp çeşitli bilgiler edindik.

VIII. Sonuç

Bu projeyi gerçekleştirerek bir web indeksleme uygulaması gerçekleştirdik. Flask frameworkü ve web teknolojilerini kullanmakta tecrübeler edindik. Doğal dil işleme hakkında teorik bilgiler edindik. Oluşturduğumuz uygulama ile frekans hesaplama, anahtar kelime bulma, benzerlik hesaplama, indeksleme ve semantic analiz gibi çeşitli işlemleri gerçekleştirdik.

IX. Kaynaklar

- https://www.geeksforgeeks.org/python-sort-python-dictionaries-by-key-or-value/ (Erişim Tarihi: 21.03.2021)
- https://www.geeksforgeeks.org/ordereddict-in-python/
- (Erişim Tarihi: 21.03.2021)
- https://www.w3schools.com/python/ref_requests_response.asp (Erişim Tarihi: 21.03.2021)
- (Erişim Tarihi: 21.03.2021)
- https://github.com/spyoungtech/grequests(Erişim Tarihi: 21.03.2021)
- https://stackoverflow.com/questions/3075550/how-can-i-get-href-links-from-html-using-python (Erişim Tarihi: 22.03.2021)
- https://anytree.readthedocs.io/en/latest/(Erişim Tarihi: 22.03.2021)
- https://pythonprogramminglanguage.com/get-links-from-webpage/ (Erişim Tarihi: 22.03.2021)
- https://www.nltk.org/(Erişim Tarihi: 23.03.2021)