KOCAELİ ÜNİVERSİTESİ

BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

YAZILIM LABORATUVARI – II

2020 – 2021 BAHAR DÖNEMİ

PROJE 1

WEB İNDEKSLEME UYGULAMASI

ÖĞRENCİ: **Ahmet YAZTÜRK**

NO: **190201175**

A. Geliştirme Ortamı

Proje Linux işletim sistemi üzerinde, Django çatısı altında, Python 3 ile geliştirilmiştir. HTML dosyasının ayrıştırılması için BeautifulSoup kütüphanesi kullanılmıştır.

B. Aşamalar

Her aşama için kullanılacak fonksiyonlar "**uygulama**" isimli dizinde kendi dosyasında tutulmaktadır. Örneğin *asama1.py*, *asama2.py* gibi. Bu dosyalarda asıl işi yapan fonksiyona **hesapla** adı verilmiştir. Ortak kullanılması düşünülen fonksiyonlar ise *parse.py* dosyasındadır. Bunlar haricinde *__init__.py* dosyasında birkaç sabit tanımlanmıştır:

- **sayfalar**: sayfalara verilecek takma isimler ve başlıkları
- **tekURL**: Aşama1 ve aşama2 için form şablonu (string)
- **ikiURL**: Aşama3 için form şablonu
- **cokURL**: Aşama4 ve aşama5 için form şablonu

stopwords.py dosyasında ise İngilizce'deki anahtar kelime niteliği taşımayan kelimelerin bir listesi tanımlanmıştır.

views.py dosyasındaki dosyalar HTTP isteklerine yanıt vermekten sorumludur. **frekans**, **anahtar**, **benzerlik**, **indeksleme** ve **semantik** isimli fonksiyonlar ilgili aşamalar için form oluştururlar ve şablon olarak *form.html* dosyasını kullanırlar. **frekans_s**, **anahtar_s**, **benzerlik_s**, **indeksleme_s** ve **semantik_s** fonksiyonları ise diğer dosyalarda tanımlı fonksiyonları kullanarak elde ettiği verilerden HTML kodu üretirler ve bu kodu **content** isimli parametreye atayarak *sonuc.html* dosyasına yerleştirirler.

1. Aşama

asama1.hesapla fonksiyonu aslında **parse.to_wordlist** fonksiyonuna bir çağrıdan ibarettir. **to_wordlist**, öncelikle **parse.to_stringlist**'i kullanarak URL'si verilen sayfanın içeriğini string listesi olarak alır. Daha sonra bu stringlerdeki bütün kelimeler tek bir listede toplanır. Ardından Python'un *collections* kütüphanesinin *Counter* sınıfı yardımıyla bu liste kelime tekrarlarının yer aldığı bir hash tablosuna (*dictionary*) dönüştürülür. *Counter.most_common* metodu ile de sıradan bir ikili listesine dönüştürülür (**[(kelime, frekans)]** şeklinde).

to_stringlist 2. aşama düşünülerek ayrı fonksiyon olarak tasarlanmıştır. Burada her bir string, bir HTML elementinin içeriğidir. Bu fonksiyonda *BeautifulSoup* kütüphanesi kullanılır. Öncelikle yine bu kütüphane yardımıyla <*script*> ve <*style*> öğeleri silinir. Ardından özel karakterler boşluğa çevrilir. Bu karakterlerin boşluğa çevrilmeyip tamamen silinmesi durumunda kelime ayrıştırmada problem olmaktadır.

2. Aşama

Bu aşama için öncelikle içeriği kısa olan HTML elementlerinde (mesela başlıklarda) yer alan kelimelerin anahtar kelime olma olasılığının daha yüksek olduğu düşünülmüştür. Bunun için her bir bölümde kelime frekanslarının bu bölümdeki toplam kelime sayısının karesine bölünmesi gibi bir yöntem izlenmiştir. Ancak bu durumda *follow, share* gibi gereksiz kelimelerin aşırı yükseldiği gözlemlenmiştir. Bu nedenle tam tersi bir yöntem izlenerek her bölümdeki kelime frekanslarının karesi alınmıştır.

- **asama2.hesapla** fonksiyonunda öncelikle **parse.to_stringlist** kullanılarak string listesi alınır. Her bir string bir HTML elementinin içeriğidir.
- Kelimelerin frekanslarını tutmak için boş bir hash tablosu (points) oluşturulur.
- Her bir string için şunlar yapılır:
  - String kelimelere bölünür
  - Boş frekans tablosu oluşturulur
  - Rastlanılan her kelime için (stopword değilse ve rakamla başlamıyorsa) frekans tablosundaki (**frequencies**) değeri bir artırılır. Bunun için **asama2.increase** fonksiyonu kullanılır. Ayrıca toplam kelime sayısını tutan **total** değişkeni de bir arttırılır.
  - Bir stringdeki kelimeler bitince oluşan **frequencies** tablosu yardımıyla **points** tablosu güncellenir. Bunun için **update** fonksiyonu kullanılır. Bu fonksiyon kısaca stringdeki frekansların karelerini alır ve ilgili kelime puanına ekler.
- Bütün stringler dolaşıldıktan sonra tamamlanan **points** tablosu, düz ikili listesine dönüştürülür ve puana göre sıralanır. *sort*'a parametre olarak aktarılan **sort_f** fonksiyonu, sıralama kriteri olarak kelimelerin değil, puanın kullanılmasını sağlar.

Hazırlayan: Ahmet Yaztürk
Kocaeli Üniversitesi, Mart 2021

Resim 1 – *Sonuçlar bölümümüzün anasayfasına aittir.*

Örneğin iki bölümden oluşan bir web sayfasında iki kelimenin 10'ar defa geçtiğini düşünelim. Birinci kelime 2+8 ise puanı 2x2 + 8x8 = 68 olacaktır. Buna karşın 5+5 defa tekrar eden ikinci kelimenin puanı 50 olacaktır. 10 farklı bölümde 1'er defa geçen kelimenin ise puanı 10 olacaktır.

3. Aşama

Benzerlik skorlaması için özetle, her iki sayfada ortak bulunan kelimelerin puanları çarpılmış, elde edilen bu çarpımlar toplanmış ve bu puan toplam kelime sayısına (her iki sayfanın toplamı) bölünmüştür. Her iki sayfanın toplam kelime sayılarının çarpımlarına bölünmesinin, küçük sayfalar için haksız avantaj yaratacağı düşünülmüş, bu nedenle çarpma yerine toplama tercih edilmiştir.

asama3.hesapla fonksiyonu iki adet (kelime,frekans) ikilileri listesi alır ve bunları iç içe iki *for* döngüsüyle hesaplar. **total** fonksiyonu ise basitçe, yine bu listeler yardımıyla sayfaların toplam kelime sayısını hesaplar.

Burada listelerde kullanılan aslında kelime frekansları değil, 2. aşama yardımıyla elde edilen puanlardır. Eğer kelime frekansları tercih edilecekse, **views.benzerlik_s** fonksiyonundaki **asama2.hesapla** çağrısı, **asama1.hesapla** olarak değiştirilmelidir.

4. Aşama

Site indeksleme aşaması için, bir URL'den elde edilen veriler ağaç yapısında tutulmuştur. Her bir düğüm **(url, [frekans listesi], [alt url ağacı])** şeklindedir. Bu ağaç **parse.recursive_wordlist** fonksiyonu ile oluşturulur. Daha sonra bu ağaç **asama4.recursive_scoring** fonksiyonu ile **(url, puan, [alt url ağacı])** şeklindeki ağaca dönüştürülür. Yani kelime frekansları puana dönüştürülür.

Alt url puanlarının ana url'yi etkilemesi için farklı bir ağırlıklandırma kullanılmıştır. Alt url sayısı N olmak üzere ana URL, (N/2 + 5) ağırlığa sahiptir. Bu durumda toplam ağırlık 3N/2 + 5 olmaktadır. Alt url'lerin ortalaması O, ana url'nin asıl puanı P ise formül:

(Px(N/2 +5) + OxN) / (3N/2 + 5) olmaktadır.

O = Toplam/N olduğuna göre OxN yerine toplam yazılabilir. Kodda bu şekilde sadeleştirilerek yazılmıştır.

Bu formüle göre alt url sayısı arttıkça ana url'nin ağırlığı %34'e kadar düşebilir. Tek alt url varsa %85 olmaktadır.

Puanları içeren ağaç oluşturulduktan sonra ayrıca sıralı puan tablosu oluşturulmaktadır (**asama4.concat** fonksiyonu ile). **views.indeksleme_s** fonksiyonundaki çağrıya hem ağaç hem tablo döndürülür. Bu sayede veriler ağaç yapısında yazdırılırken sıralaması da yazdırılabilir. **views.agac_yazdir** yine rekürsif yapıda bir fonksiyondur ve HTML kodu üretir.

Derinlik 3 olduğunda, alt url sayısına bağlı olarak bu aşama oldukça geç yanıt vermektedir. Bu yüzden derinliğin 2 yapılması tavsiye edilir.

Resim 2 – *Derinlik 2 yapılarak ve URL kümesi alanına 2 URL girilerek alınan sonuç*

5. Aşama

Bu aşama gerçeklenmemiştir.