|  |
| --- |
|  |
| Algoritma Analizi |
|  |
| 2. Ödev: Dinamik Programlama |

**Algoritma analizi**

29 Ekim 2017

Yazan: Emrehan İLHAN

Algoritma Analizi

2. Ödev: Dinamik Programlama

Yöntem

### Problem

Fazla sayıda kelime içeren bir sözlük, hashing kullanarak depolanacak. Daha sonar verilen bir kelime hash mantığı kullanılarak bu sözlükte aranacak. Kelime var ise hash sayesinde kısa sürede bulunacak, yok ise sözlükteki tüm kelimeler ile aradığımız kelime arasında Edit Distance hesaplanacak.

## Çözüm

Problemi 3 aşamada inceleyebiliriz. Sözlüğü hash ile saklama ve bu sözlükte arama yapma, 2 kelimeyi edit distance yöntemi ile uzaklığını bulma, backtracking ile iki kelimenin birbirine çevirme.

Hash problem için sözlüğün boyutunun **iki katına en yakın asal sayı** uzunluğunda bir dizi tutuyoruz. C dilindeki struct yapısı ile bu dizinin her elemanı [kelime, boş mu] elemanlarını tutuyor.Bir kelime dosyadan okunduğunda, bu kelimenin harfleri ile bir indis değeri belirleniyor. Bu indis eğer boşsa kelime buraya ekleniyor, boş değilse boşluk bulunana kadar indis arttırılıyor. Bu adımlarla oluşan hash yapısını dosyada tutarak sürekli yapmaktan kurtuluyoruz.

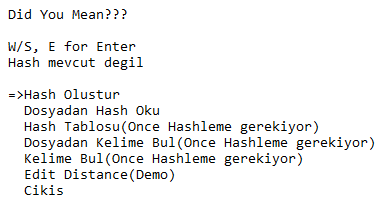
Hash üzerinde bir kelimeyi ararken, tahmin ettiğimiz hash değerini hesaplıyoruz ve o göze bakıyoruz. Göz boşsa kelime sözlükte yoktur, doluysa indisi boş göz yada aradığımız kelimeyi bulana kadar arttırıyoruz.

Aradığımız kelime sözlükte bulunmuyorsa ona en benzeyen kelimeyi önermemiz gerekiyor. Bunu yapmak için aradığımız kelimelerle, sözlükteki tüm kelimeleri edit distance yöntemi ile birbirine olan uzaklığını hesaplanır. Aradığımıza en yakın kelimenin indisi saklanır ve backtrack’e gönderilir.

Elimizde Edit Distance matrisi olan iki kelimeye backtracking yapmak için; son sütun ve son satır elemanından başlanır. O göze hangi gözden gelindiyse, o gözün değerine ve karşılaştırılan harflerin değerine göre ekleme, silme, değiştirme işlemi varsa bu işlem yakalanır.

Uygulama

## Programın Hazırlanması



Hash Oluştur: Kelimeleri içeren text dosyasından, kelimeleri okur ve hash oluşturur. Oluşan hashi bir text dosyasına yazdırır.

Dosyadan Hash Oku: Hash Oluştur’un oluşturmuş olduğu text dosyasından hazırlanmış hash’i okur

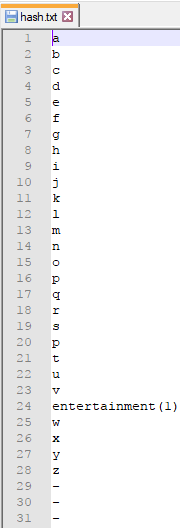
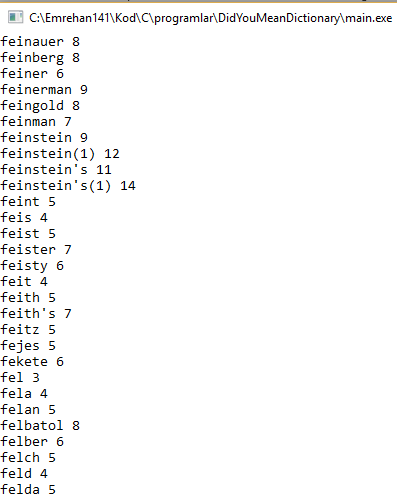
Hash Tablosu: Hash’i yazdırır.

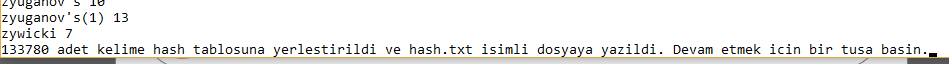
Dosyadan Kelime Bul: Ödevde verilen test.txt dosyasındaki kelimeleri sözlükte arar bulamazsa kelime önerir.

Kelime Bul: Verilen kelimeyi sözlükte arar, bulamazsa kelime önerir.

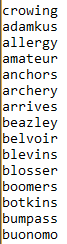
Edit Distance: Edit Distance yöntemini 2 kelime üzerinde dener.

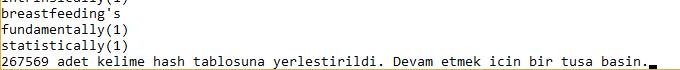
# Hash Oluştur



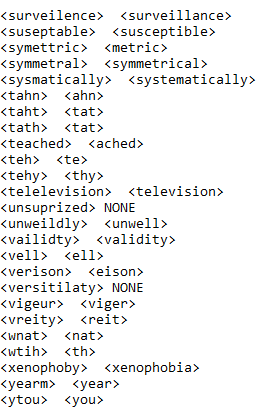
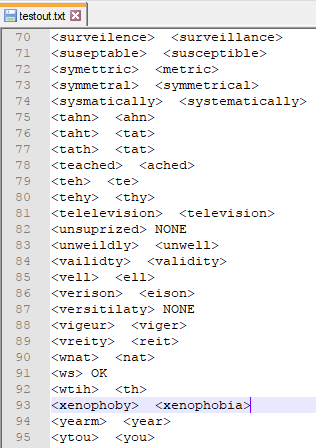


## Dosyadan Hash Oku

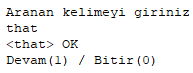


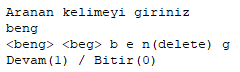


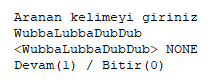
## Dosyadan Kelime Oku

## Kelime Bul







Ödevde İstenilen 10 Kelime

|  |  |  |
| --- | --- | --- |
| <verison> | <eison> | v(delete) e r(delete) i s o n |
| <versitilaty> | NONE |  |
| <vigeur> | <viger> | v i g e u(delete) r |
| <vreity> | <reit> | r e i t y(delete) |
| <wnat> | <nat> | w(delete) n a t |
| <ws> | OK |  |
| <wtih> | <th> | w(delete) t i(delete) h |
| <xenophoby> | <xenophobia> | x e n o p h o b i(insert) a(change) |
| <yearm> | <year> | y e a r m(delete) |
| <ytou> | <you> | y t(delete) o u |

Sonuç

## Karmaşıklık Analizi

**Hash oluşturma: N**   
Hash oluştururken karmaşıklık sözlükteki kelime sayısı kadar olur

**Hash arama yapma: 1**   
Hash yapısında kelime aramanın karmaşıklığı 1’dir.

**Edit Distance: NxM**   
İki kelimenin uzunluğunun çarpımıdır.

**Bu programda Edit Distance İle Kelime Önerme: NxU1xU2 => N olur**  
(U1: Aranan kelime uzunluğu, U2: Sözlükte bakılan kelime uzunluğu, N Sözlükte eleman sayısı )