YILDIZ TEKNİK ÜNİVERSİTESİ

ELEKTRİK-ELEKTRONİK FAKÜLTESİ

BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

BLM-1012—YAPISAL PROGRAMLAMAYA GİRİŞ FİNAL PROJESİ

BOYER MOORE ALGORİTMASI

Ders Yürütücüsü
Mehmet Fatih Amasyalı
Haziran,2021

Öğrenci: Selahattin Yasin ÇAYCI
No: 20011099
E-posta: yasin.cayci@std.yildiz.edu.tr

İÇERİK

VİDEO ADRESİ

https://drive.google.com/file/d/1QMBKiiV1YBp_ng0y0m6i9f9-pkReF_Gf/view?usp=sharing

Boyer Moore Algoritması Nedir?

Boyer Moore algoritması bir kelimenin (ya da bir metin parçasının) bir metin içerisinde aranmasını sağlayan algoritmadır. 1977'de Robert S. Boyer ve J Strother Moore tarafından geliştirilmiştir. Basitçe bu algoritmada bir kelimenin aranan metinde bakılması ve bakıldığı yerde bulunamaması durumunda nerede olabileceği ile ilgili bir bilginin elde edilmesi hedeflenir.

ALGORİTMANIN TEMEL AMACI

Bu algoritmadaki amaç bütün harfleri teker teker kontrol eden doğrusal aramadan (linear search) daha iyi bir sonuç elde etmektir.

NASIL ÇALIŞIR?

Algoritmada aranan dizgiye pattern(desen), aramanın yapılacağı dizgiye text(metin) denilmektedir. Bu algoritmanın tanıtılmasından önce, metin içinde arama yapmanın klasik yolu, metnin her bir karakterini desenin ilk karakteri için incelemektir. Eşleşme bulunduğunda, metnin sonraki karakterleri, desenin karakterleriyle karşılaştırılır. Eğer eşleşme olmazsa, metin bir eşleşme bulmak için tekrar karakter kontrol edilir. Bu nedenle metindeki hemen hemen her karakterin incelenmesi gerekir.

Bu algoritmadaki temel fikir, eğer pattern in sonu text ile karşılaştırılırsa, text in her karakterini kontrol etmek yerine text boyunca atlamalar yapılabileceğidir. Bunun işe yaramasının nedeni, pattern i text e göre sıralarken, pattern in son karakterinin metindeki son karakterle karşılaştırılmasıdır. Karakterler eşleşmiyorsa, text boyunca geriye doğru aramaya devam etmeye gerek yoktur. Text üzerinde pattern in uzunluğu kadar atlama yapılır ve eşleştirme işlemlerinde hız sağlanır.

ÖRNEK:

```
0 1 2 3 4 5 6 7 8 9 ...
a b b a d a b a c b a
b a b a c
          b a b a c
```

İlk karşılaştırma olan 4. Sütundaki d-c karşılaştırmasında uyumsuzluk var. Text teki d pattern de yer almıyor. Bu yüzden pattern in 0 - 4 arasındaki sütunlarla eşleşmesi imkansız, pattern i kendi uzunluğu kadar kaydırarak 5.sütuna getiriyoruz ve büyük bir hız kazancı sağlıyoruz.

Bu algoritmada kaydırma işleminden önce preprocessing adımı uygulanır. Preprocessing adımı uyumsuzluk durumlarında kaç karakter kaydırılacağını belirler. Bunu da iki adet tabloya göre yapar. Bunlar bad character table ve good suffix tir.

Bad Character Table Oluşturma

Pattern deki harfler tekrar etmeyecek şekilde yazılır. En sağa ise * sembolü yazılır. * sembolünün değeri pattern in uzunluğudur. Eğer ki textteki karşılaştırılan harf pattern in içinde yer almıyorsa * ın değeri kadar kaydırma yapılır (üstteki örnekteki gibi). Diğer harflerin kaydırma sayısı ise şu formüle göre bulunur:

Pattern uzunluğu- mevcut index -1

Tablodaki son karakter bu formülden bağımsız olarak her zaman 1 değerine sahiptir.

ÖRNEK:

0 1 2 3 4 5 6 7

GCAGAGAG

Formül: Uzunluk – index - 1

Uzunluk = 8

G = 8 – 0 – 1 = 7

C = 8 – 1 – 1 = 6

A = 8 – 2 – 1 = 5

G = 8 – 3 – 1 = 4

A = 8 – 4 – 1 = 3

G = 8 – 5 – 1 = 2

A = 8 – 6 – 1 = 1

G = 8 – 7 - 1 = 0 ama son karakter olduğu için 1

A	C	G	*
1	6	1	8

Good Suffix Oluşturma

Bu yöntemi internetteki anlatımların doğru çalışmaması sebebiyle yapmadım. Durumu Fatih Amasyalı hocama ilettim, sadece bad match table ı yapmamın yeterli olacağı yanıtını aldım. Kaydırma işlemlerini bad match table a göre yaptım.

ARAMA İŞLEMLERİ

Örnek olarak text HERE IS A SIMPLE EXAMPLE olsun

Pattern ise EXAMPLE olsun

Öncelikle bad character table oluşturulur.

Uzunluk = 7

E = 7 – 0 – 1 = 6

X = 7 – 1 – 1 = 5

A = 7 – 2 – 1 = 4

M = 7 – 3 – 1 = 3

P = 7 – 4 – 1 = 2

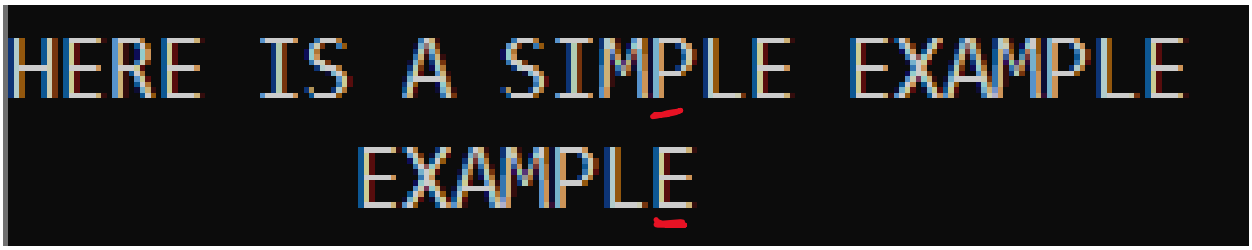
L = 7 – 5 – 1 = 1

E son karakter, 1

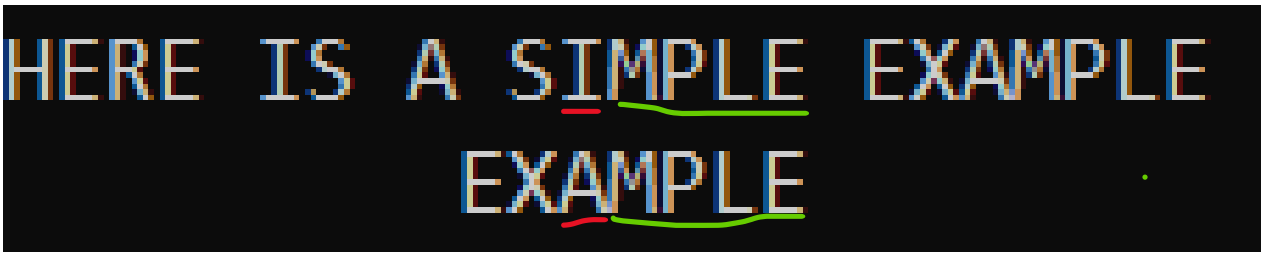
```
----------------------------------------
Bad Character Table
  E     X     A     M     P     L     E     *
  1     5     4     3     2     1     1     7
----------------------------------------
```

```
HERE IS A SIMPLE EXAMPLE
EXAMPLE
```

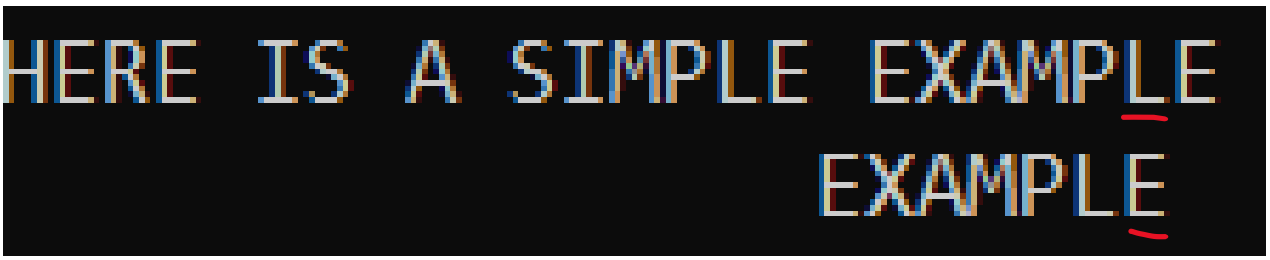
S ile E eşleşmez. S nin tablodaki değeri * yani 7 dir, 7 birim kayma olur.



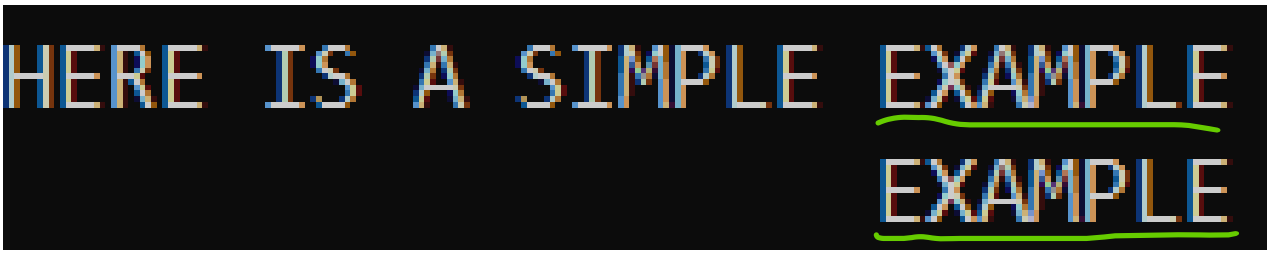
P ile E eşleşmez. P nin tablodaki değeri 2 dir, 2 birim kayma olur.



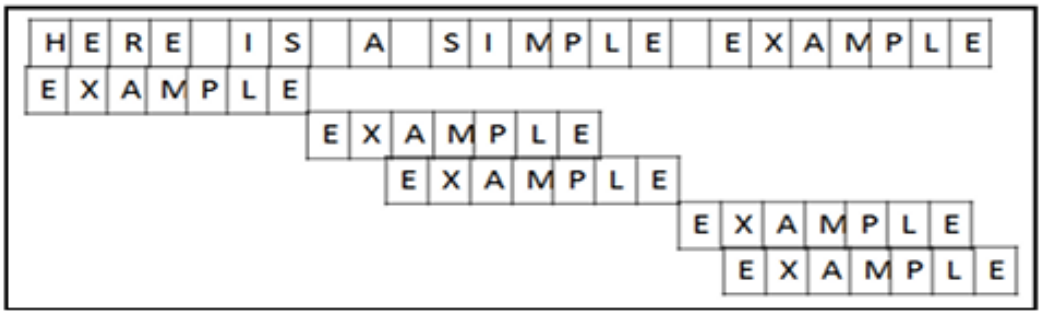
I ile A eşleşmez. I nin tablodaki değeri * yani 7 dir, 7 birim kayma olur.



L ile E eşleşmez. L nin tablodaki değeri 1 dir, 1 birim kayma olur.



Tüm harfler eşleşir. Text in sonuna gelindiği için arama bitmiştir.



Kullanım Yerleri

Uzun metinlerde çok avantajlı olduğundan dolayı kullandığımız çoğu metin editöründe kullanılır.

Big-Data, Database searching

AVANTAJLAR

Üzerinde arama yapılan yer büyüdükçe hızlanması

Arama yönüne bağlı olarak metin aramasını sağdan sola doğru yapması

Pre-process kısmına pattern i sokması

DEZAVANTAJLAR

Kısa metinlerde rakip algoritmalardan daha yavaş çalışması

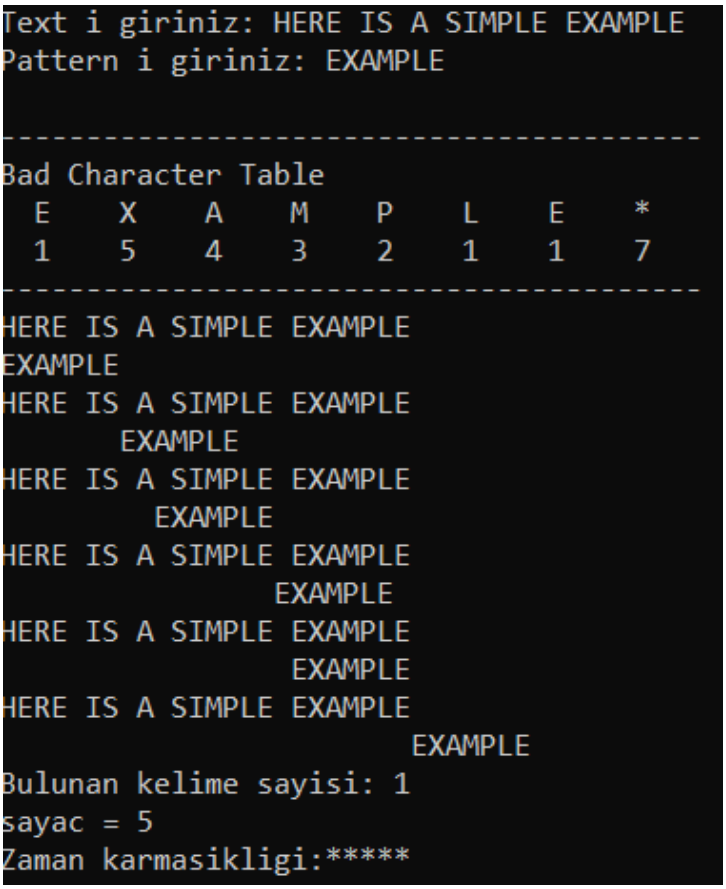
Arama yönüne bağlı olarak metin aramasını sağdan sola doğru yapması

ARAMA ZAMANI KARMAŞIKLIĞI

Algoritma performansı doğrusal aramada aranan kelime uzunluğu (m) ile hedef kelime uzunluğu (n) çarpımıdır: m*n

Ancak BM algoritması burada devreye girerek aranan kelimenin bütün harflerinin kontrolünü her seferinde engellemektedir. Bu yüzden algoritma başarısı best case durumunda O(n/m), worst case durumunda doğrusal hıza sahip olup O(n) ile ifade edilebilir. Ortalama olarak ise O(n/m) den biraz fazladır

ÖRNEK:



SINIRLARI

#define SIZE kısmındaki sayı

RAKİPLERİ

BOYER MOORE vs BRUTE FORCE

Brute Force algoritması bir text içinde pattern arama algoritmalarından bir tanesidir.

Algoritma aramaya ilk karakterden başlar ve karakter eşleşmeleri sağlandıkça devam eder. Yani ön ek eşleştirme algoritmasıdır.

Pencerenin tam karakter eşleşmesi veya eşleşmemesi durumlarında pencereyi tam olarak 1 pozisyon sağa kaydırarak arama işlemi sürdürülür.

Tam eşleşme durumunda aranan sonuç bulundu kabul edilir, arama işlemi kaldığı yerden devam eder. Pattern penceresi, textin sonuna geldiğinde algoritma sonlanır.

Sonuç olarak bu algoritmada hedef pattern, text içinde lineer olarak aranmış olur.

Brute Force algoritması O(M*N) iken Boyer Moore algoritması O(N/M)'e yakın bir zaman karmaşıklığına sahiptir. Bundan dolayı genel anlamda Boyer Moore algoritması Brute Force'a göre çok daha verimli ve tasarruflu bir algoritma olarak öne çıkar ve asıl çıkış sebebi de bilinen doğrusal aramaya karşın yeni ve daha efektif bir yöntem arayışından gelmektedir.

Brute Force alanında son derece verimsiz bir algoritmadır ve bundan dolayı gündelik hayatta işimizi kolaylaştırabilecek uygulama alanları içinde pek yer almaz. Öte yandan çoğunlukla doğrudan deneme-yanılma tabanlı sanal operasyonlarda kullanılmaktadır.

Table1.Table of findings for execution time(in seconds) based on various inputs for different algorithms.

Algorithm	Length of I/P No. of executions	Nature of I/P									Findings
		AU			AL			M			
		1	2	3	1	2	3	1	2	3	
Brute Force (BFA)	Short	6	4	3	5	4	3	5	3	2	BFA is good for short strings and search is faster for lower case than uppercase letters.
	Long	8	5	3	8	7	5	9	7	6	
Karp Rabin (KRA)	Short	6	5	4	4	3	2	4	3	2	KRA results are similar to BFA but and search is faster for mixed letters compared to BFA.
	Long	8	6	5	8	7	6	9	6	5	
Boyer Moore (BMA)	Short	7	7	5	6	6	6	4	3	3	BMA is good for long strings than the short.
	Long	6	5	4	5	4	4	5	4	3	
AU- All Uppercase					AL- All Lowercase						M- Mixed

BOYER MOORE vs KMP

Bu algoritma ve Boyer Moore'un arasında işlemsel ve akış olarak pek çok farklılık bulunsa da ikisi de aynı amaca hizmet ettiğinden ötürü birbirlerine üstünlük sağladığı özelliklere bağlı olarak günlük hayatımızda belli alanlarda her ikisi de aktif olarak kullanılmaktadır.

BOYER MOORE ALGORİTMASININ KMP'YE GÖRE AVANTAJLARI

Büyük boyutlu karakter kümeleri üzerinde yapılan algoritmalarda mantıksal doğası gereği Boyer-Moore, KMP'ye göre daha efektif ve kullanışlıdır. Bundan dolayı Big-Data, Database searching gibi büyük çaplı metin arama işlemlerinde Boyer-Moore çok daha yaygın olarak kullanılır.

Arama yönünden dolayı sağdan yapılan metin aramalarında Boyer-Moore genellikle daha avantajlı ve başarılıdır.

Belirli bir alt karakter kümesini kendi içerisinde çok kez barındıran metinlerde Boyer-Moore KMP'den daha iyi arama performansı gösterir. Bu tarz çoklu ve belirli tipte karakter kümelerinden oluşan büyük kümelerin aranmasının yanı sıra karakter farklılığı ve varyantı daha fazla olan kümelerde yine Boyer-Moore daha başarılı ve tercih edilen bir algoritmadır.

KMP ALGORİTMASININ BOYER-MOORE'A GÖRE AVANTAJLARI

Tekrar eden karakter kümelerinde KMP, Boyer-Moore'dan çoğu zaman için daha iyi arama performansı gösterir. Bundan dolayı belli türde karakterlerin bir araya getirdiği tekrarlı dizgilerin aranması (DNA içerisinde gen parçasının aranması gibi) işlemlerde KMP algoritması çok daha yaygın kullanılır.

Aynı zamanda KMP soldan sağa doğru arama yaparken Boyer-Moore sağdan sola doğru arama yapar ve arama yönüne bağlı olarak soldan yapılan metin aramalarında KMP genellikle daha avantajlı ve başarılıdır.

Öte yandan mantıksal doğası gereği KMP algoritması küçük boyutta karakter kümeleri üzerinde yapılan aramalarda Boyer-Moore'dan daha başarılıdır.

Tablo 2: Alfabeler için süre değerleri (s)

Yöntem		DNA			RAKAM			DOĞAL DİL		
		Kısa	Orta	Uzun	Kısa	Orta	Uzun	Kısa	Orta	Uzun
	Zhu-Takaoka	7,66	5,82	7,99	4,21	3,00	4,24	2,95	1,26	1,45
	Berry Ravindran	8,89	7,79	10,72	4,79	2,82	3,79	3,57	1,33	1,73
	Horspool	9,33	11,93	14,26	4,09	3,84	4,87	2,40	1,34	1,47
	Boyer Moore	11,39	8,10	11,78	5,76	4,93	6,36	3,35	1,73	1,70
	Turbo Boyer Moore	13,49	9,02	7,96	7,47	5,62	6,86	4,15	1,79	1,77
	Tuned Boyer Moore	7,89	13,79	20,58	3,20	5,07	8,35	2,59	1,48	1,55
	Raita	9,39	14,88	20,80	4,24	5,55	8,36	2,40	1,36	1,45
	Smith	13,06	16,02	19,36	5,44	4,45	6,43	3,90	1,70	1,65
	Quick Search	14,66	17,46	20,92	5,16	4,87	6,65	2,75	1,54	1,58
	Morris Pratt	28,05	16,18	10,75	19,76	17,35	15,21	15,37	15,40	14,85
	Apostolico-Crochemore	25,33	11,89	5,99	24,84	19,48	13,21	22,72	23,12	23,21
	Brute Force	30,51	33,68	36,66	17,07	18,63	21,46	12,73	12,67	11,93
	Knuth Morris Pratt	32,59	21,28	15,82	24,94	22,43	20,17	20,61	20,70	19,81

EKRAN ÇIKTILARI

```
Text i giriniz: GAESADGAEGAFAFSSFAFBDGAEDABDGDAGAEBSSGAESAFGAEASFAE
Pattern i giriniz: GAE

----------------------------------------
Bad Character Table
  G    A    E    *
  2    1    1    3
----------------------------------------
GAESADGAEGAFAFSSFAFBDGAEDABDGDAGAEBSSGAESAFGAEASFAE
GAE
GAESADGAEGAFAFSSFAFBDGAEDABDGDAGAEBSSGAESAFGAEASFAE
   GAE
GAESADGAEGAFAFSSFAFBDGAEDABDGDAGAEBSSGAESAFGAEASFAE
      GAE
GAESADGAEGAFAFSSFAFBDGAEDABDGDAGAEBSSGAESAFGAEASFAE
         GAE
GAESADGAEGAFAFSSFAFBDGAEDABDGDAGAEBSSGAESAFGAEASFAE
            GAE
GAESADGAEGAFAFSSFAFBDGAEDABDGDAGAEBSSGAESAFGAEASFAE
               GAE
GAESADGAEGAFAFSSFAFBDGAEDABDGDAGAEBSSGAESAFGAEASFAE
                  GAE
GAESADGAEGAFAFSSFAFBDGAEDABDGDAGAEBSSGAESAFGAEASFAE
                     GAE
GAESADGAEGAFAFSSFAFBDGAEDABDGDAGAEBSSGAESAFGAEASFAE
                        GAE
GAESADGAEGAFAFSSFAFBDGAEDABDGDAGAEBSSGAESAFGAEASFAE
                           GAE
GAESADGAEGAFAFSSFAFBDGAEDABDGDAGAEBSSGAESAFGAEASFAE
                              GAE
GAESADGAEGAFAFSSFAFBDGAEDABDGDAGAEBSSGAESAFGAEASFAE
                                 GAE
GAESADGAEGAFAFSSFAFBDGAEDABDGDAGAEBSSGAESAFGAEASFAE
                                    GAE
GAESADGAEGAFAFSSFAFBDGAEDABDGDAGAEBSSGAESAFGAEASFAE
                                       GAE
GAESADGAEGAFAFSSFAFBDGAEDABDGDAGAEBSSGAESAFGAEASFAE
                                          GAE
GAESADGAEGAFAFSSFAFBDGAEDABDGDAGAEBSSGAESAFGAEASFAE
                                             GAE
GAESADGAEGAFAFSSFAFBDGAEDABDGDAGAEBSSGAESAFGAEASFAE
                                                GAE
Bulunan kelime sayisi: 6
sayac = 30
Zaman karmasikligi:******************************
```

```
Text i giriniz: ytu Dunyanin Yedi harikasindan biri
Pattern i giriniz: in

----------------------------------------
Bad Character Table
  I    N    *
  1    1    2
----------------------------------------
YTU DUNYANIN YEDI HARIKASINDAN BIRI
IN
YTU DUNYANIN YEDI HARIKASINDAN BIRI
  IN
YTU DUNYANIN YEDI HARIKASINDAN BIRI
    IN
YTU DUNYANIN YEDI HARIKASINDAN BIRI
      IN
YTU DUNYANIN YEDI HARIKASINDAN BIRI
        IN
YTU DUNYANIN YEDI HARIKASINDAN BIRI
          IN
YTU DUNYANIN YEDI HARIKASINDAN BIRI
            IN
YTU DUNYANIN YEDI HARIKASINDAN BIRI
              IN
YTU DUNYANIN YEDI HARIKASINDAN BIRI
                IN
YTU DUNYANIN YEDI HARIKASINDAN BIRI
                  IN
YTU DUNYANIN YEDI HARIKASINDAN BIRI
                    IN
YTU DUNYANIN YEDI HARIKASINDAN BIRI
                      IN
YTU DUNYANIN YEDI HARIKASINDAN BIRI
                        IN
YTU DUNYANIN YEDI HARIKASINDAN BIRI
                          IN
YTU DUNYANIN YEDI HARIKASINDAN BIRI
                            IN
YTU DUNYANIN YEDI HARIKASINDAN BIRI
                              IN
YTU DUNYANIN YEDI HARIKASINDAN BIRI
                                IN
YTU DUNYANIN YEDI HARIKASINDAN BIRI
                                  IN
YTU DUNYANIN YEDI HARIKASINDAN BIRI
                                    IN
Bulunan kelime sayisi: 2
sayac = 21
Zaman karmasikligi:**********
```

C KODU

20011099.c

```c
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#define SIZE 1000
#define totalAsciiNumber 256 // ASCII de toplamda 256 karakter var

void badCharacterRule(char pattern[SIZE], int patlen, int badCharacterTable[totalAsciiNumber]){
    int i;
    for (i = 0; i < totalAsciiNumber; i++) // Tüm ascii karakterleri için atlama sayısı patlen oldu
        badCharacterTable[i] = patlen;

    for (i = 0; i < patlen; i++){ // bad character rule a göre tablonun formülü length-i-1
        badCharacterTable[(int)pattern[i]] = patlen-i-1;
        if(patlen-i-1 == 0)
            badCharacterTable[(int)pattern[i]] = 1; // son harf her zaman 1 olur
    }
}

// Boyer Moore Arama Algoritması
void search(char text[SIZE], char pattern[SIZE]){
    int patlen = strlen(pattern);
    int textlen = strlen(text);
    int badCharacterTable[totalAsciiNumber];

    badCharacterRule(pattern, patlen, badCharacterTable); // Ön işlem yapılır

    int shift = 0,i,found = 0,counter = 0;
    while (shift <= (textlen - patlen)){ // Text in sonuna kadar kaydırma işlemi yapılır
        int j = patlen - 1;
        while (j >= 0 && pattern[j] == text[shift + j]) // Text ve pattern tamamen eşleşene kadar j yi azaltır
            j--; // Tamamen eşleşme olursa j -1 e kadar azalır

        if (j == -1) {
            found++;
            counter += patlen; // karşılaştırma sayısı patlen kadar artmış olur
            shift += patlen; // eşleşme durumunda birer birer ilerlemek yerine tek seferde pattern in uzunluğu kadar ilerlemek gerekir
            if(shift <= textlen){
                printf("%s\n",text);
                for(i = 0; i < shift; i++)
                    printf(" ");
            printf("%s\n",pattern);
            }
        }

        else{ // eşleşmeme durumu gerçekleşmiş, text in pattern ile eşleşmeyen harfinin bad character tablosundaki değeri kadar kaydırma yapmalıyız
            counter += patlen - j; // karşılaştırma sayısı yanlış karşılaştırma çıkana kadar artmış olur
            shift += badCharacterTable[text[shift + j]];
            if(shift <= textlen){
                printf("%s\n",text);
                for(i = 0; i < shift; i++)
                    printf(" ");
            printf("%s\n",pattern);
            }
        }

        if(shift > (textlen - patlen)){ // arama kısmı biterse ekrana karmaşıklığı bastırmak için
            printf("Bulunan kelime sayisi: %d\n",found);
            printf("sayac = %d\nZaman karmasikligi:",counter);
            for(i = 0; i < counter; i++)
            printf("*");
        }
    }
}
```

```
64
65  char upper(char ch[SIZE], int number){ // arama esnasındaki küçük büyük harf hassasiyeti için
66      int i;
67      for(i = 0; i < number; i++){
68          if(ch[i] >= 'a' && ch[i] <= 'z')
69              ch[i] -= 32;
70      }
71      return *ch;
72  }
73
74  void printBadChTable(char pattern[SIZE], int patlen, int badCharacterTable[SIZE]){
75      int i;
76      badCharacterRule(pattern, patlen, badCharacterTable);
77      printf("\n----------------------------------------\n");
78      printf("Bad Character Table\n");
79      for(i = 0; i < patlen; i++)
80          printf("  %c  ",pattern[i]);
81      printf("  *  ");
82      printf("\n");
83      for(i = 0; i < patlen; i++)
84          printf("  %d  ", badCharacterTable[pattern[i]]);
85      printf("  %d  ",patlen);
86      printf("\n----------------------------------------\n");
87  }
88
```

```
89  int main(){
90
91      int badCharacterTable[SIZE];
92      char text[SIZE],pattern[SIZE];
93
94      printf("Text i giriniz: ");
95      gets(text);
96      int textlen = strlen(text);
97      upper(text,textlen);
98
99      printf("Pattern i giriniz: ");
100     gets(pattern);
101     int patlen = strlen(pattern);
102     upper(pattern,patlen);
103
104     printBadChTable(pattern,patlen,badCharacterTable);
105     printf("%s\n%s\n",text,pattern);
106     search(text, pattern);
107
108     return 0;
109 }
110
```

KAYNAKLAR

1) https://en.wikipedia.org/wiki/Boyer%E2%80%93Moore_string-search_algorithm
2) https://www.inf.hs-flensburg.de/lang/algorithmen/pattern/bmen.htm
3) http://www-igm.univ-mlv.fr/~lecroq/string/node14.html
4) https://bilgisayarkavramlari.com/2009/05/19/boyer-moore-dizgi-arama-algoritmasi-boyer-moore-string-search/#perf
5) https://www.youtube.com/watch?v=J210DiuHIdU&t=132s
6) https://www.youtube.com/watch?v=c1LyW_8zPdA&t=34s
7) https://www.youtube.com/watch?v=2ZPo4Qr2tIo
8) chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/https://www.emo.org.tr/ekler/113fa9dacd22490_ek.pdf
9) chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/http://www.ijltet.org/journal/147905372037.pdf