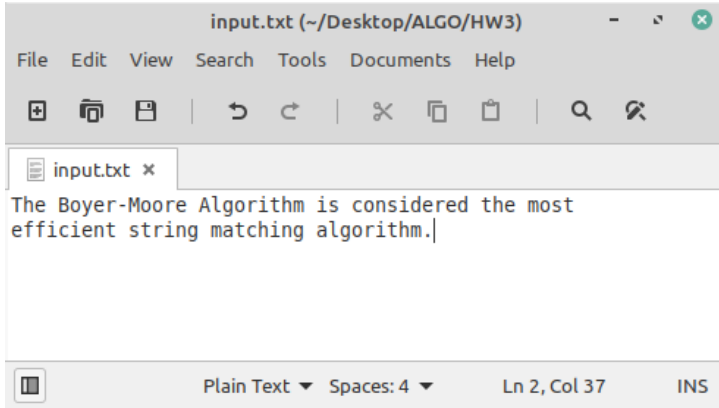


2020 Bahar Dönemi Veri Yapıları ve Algoritmalar Dersi

3. Ödev Raporu

Dokümanda çıktısı istenen örnekler:

Örnek 1:



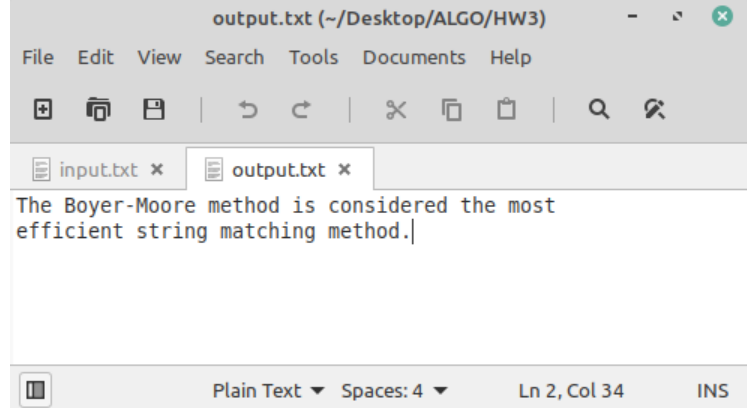
input.txt (~/Desktop/ALGO/HW3)

File Edit View Search Tools Documents Help

input.txt x

The Boyer-Moore Algorithm is considered the most efficient string matching algorithm.

Plain Text Spaces: 4 Ln 2, Col 37 INS



output.txt (~/Desktop/ALGO/HW3)

File Edit View Search Tools Documents Help

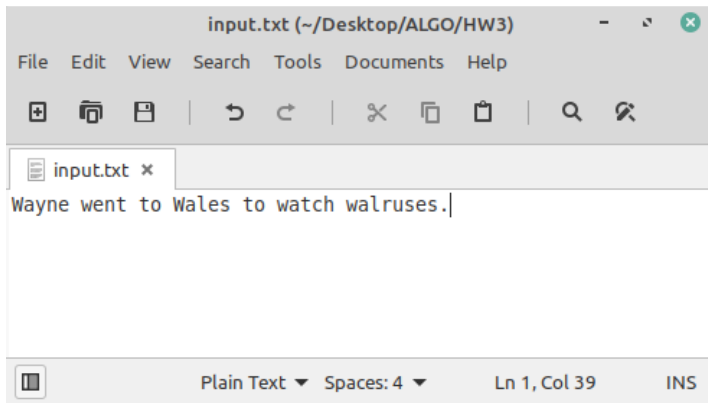
input.txt x output.txt x

The Boyer-Moore method is considered the most efficient string matching method.

Plain Text Spaces: 4 Ln 2, Col 34 INS

```
-----  
Write the name of txt file: input.txt  
-----  
Find: algorithm  
Replace: method  
Case Sensitive (yes or no):no  
-----  
Process executed in 0.000420 seconds or 0.420000 milliseconds.  
2 words were found and replaced.  
-----  
Write the name of output file: output.txt
```

Örnek 2:



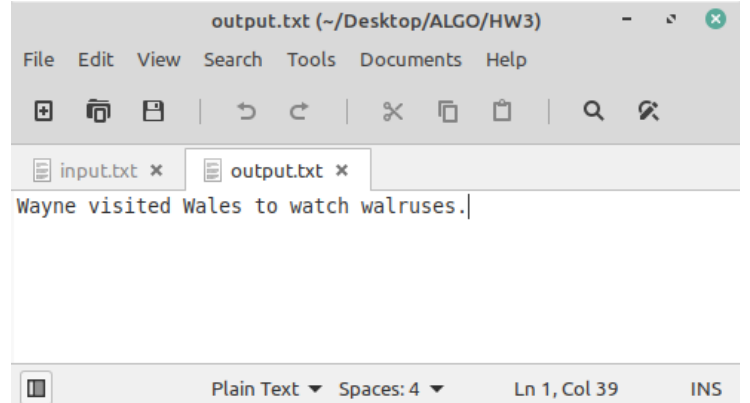
input.txt (~/Desktop/ALGO/HW3)

File Edit View Search Tools Documents Help

input.txt x

Wayne went to Wales to watch walruses.

Plain Text Spaces: 4 Ln 1, Col 39 INS



output.txt (~/Desktop/ALGO/HW3)

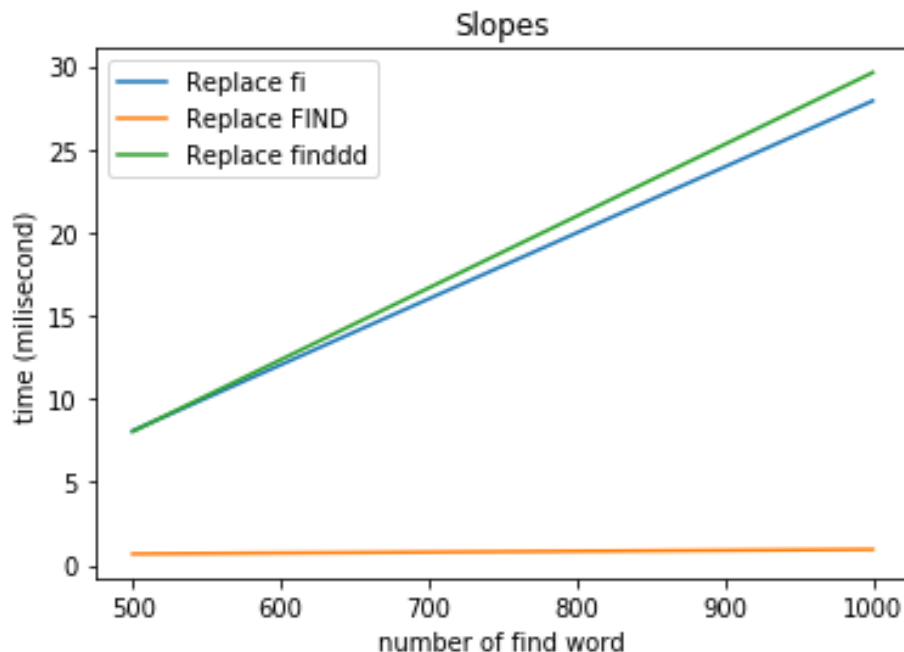
File Edit View Search Tools Documents Help

input.txt x output.txt x

Wayne visited Wales to watch walruses.

Plain Text Spaces: 4 Ln 1, Col 39 INS

```
-----  
Write the name of txt file: input.txt  
-----  
Find: went to  
Replace: visited  
Case Sensitive (yes or no):yes  
-----  
Process executed in 0.000486 seconds or 0.486000 milliseconds.  
1 words were found and replaced.  
-----  
Write the name of output file: output.txt
```



Ödevde ait C kodu:

(Algoritma ile ilgili gerekli açıklamalar yorum satırlarında yapılmıştır.)

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#include<time.h>
#define MAX 10000 //Text'in maximum uzunluğunu tanımlar.
```

//Dosyayı text dizisine okuduğumuz fonksiyon.

```
void read_txt(char text[MAX]){

    char name[15];
    printf("Write the name of txt file: ");
    scanf("%s",name);

    FILE *fp = fopen(name,"r");

    int i = 0;
    //Dosya sonuna kadar karakter karakter okuma yapar.
    while(!feof(fp)){
        text[i] = getc(fp);
        i++;
    }
    //Dizinin son elemanına string olması hasebi ile \0 koyar.
    text[i-1] = '\0';
    fclose(fp);
}
```

//Aynı dosyaya yazma işlemini yaptığımız fonksiyon.

```
void write_txt(char text[MAX]){

    char name[15];
    printf("Write the name of output file: ");
    scanf("%s",name);
    FILE *fp = fopen(name,"w");
    if(!fp){
        printf("File cannot be created!\n");
        exit(1);
    }
    //Fputs ile tek adımda bütün dizi dosyaya yazılır.
    fputs(text,fp);
    fclose(fp);
}
```

//Search fonksiyonunda kullanacağımız skip table oluşturulur.

void shiftTable(char *find, int shift_table[256], char *caseSen){

int i, size = strlen(find);

//Bulunacak kelimenin boyutuna kadar gidilerek

//tablo oluşturulur. (Kodu zaten derste yazmıştık.)

for(i = 0 ; i < 256 ; i++){

shift_table[i] = -1;

}

for(i = 0 ; i < size ; i++){

shift_table[(int)find[i]] = i;

if(!strcmp(caseSen,"no")){ //Case sensitive kapalı ise örnek olarak 'a'

indisine koyduğumuz değeri 'A' indisine de koydum.

shift_table[(int)find[i] - 32] = i;

}

}

}

//Boyer-Moore Horspool algoritması kullanarak verilen bir kelimeyi bulan fonksiyon.

int search(char *text, char *find, int shift_table[256], int index, char *caseSens){

int i, j, skip = 0, N = strlen(text), M = strlen(find);

//Aramaya her defasında 0'dan başlamak yerine bir önceki bulunan indisin bir sonrasından başlanır.

//ileride tanıtacağım exIndex değişkeninin buraya parametre olarak gönderilme sebebi bu optimizasyondur.

for(i = index + 1 ; i < N-M ; i += skip){

j = M - 1;

//Parametre olarak gönderilen caseSens değeri eğer no ise while condition ona göre düzenlenir.

if(!strcmp(caseSens,"no")){

while((j >= 0) && (find[j] == text[i+j] || find[j] - text[i+j] == 32 || text[i+j] - find[j] == 32)){

j--;

}

} //yes ise while condition ona göre düzenlenir.

else{

while((j >= 0) && (find[j] == text[i+j])){

j--;

}

}

//Tabloya bakarak ne kadar atlayacağımızı buluruz.

if(j >= 0){

skip = j - shift_table[text[i + j]];

```

        if(skip < 0){ //Eğer uyuşmuyorsa -1 değeri gelir
            skip = 1; //Cursor'ı bir adım ileri taşımak için -1 pozitif yapılır.
        }
    }
    else{
        return i; //Aranan değer bulundu ise bulunan indis döndürülür.
    }
}
return -1; //Bulunamadı ise -1 döndürülür.
}

```

//İndisi parametre olarak yer değiştirme işleminin yapıldığı fonksiyon.

```
void replace_word(char *text, char *find, char *replace, int index){
```

```
    int i;
```

```
    int lenFind = strlen(find);
```

```
    int lenReplace = strlen(replace);
```

```
    int diff = lenFind - lenReplace;
```

```
    int lenText = strlen(text);
```

```
    //Replace ile find değişkenlerinin boyutu eşit ise direkt üzerine yazılır.
```

```
    //Herhangi bir kaydırma işlemi yapılmaz.
```

```
    if(lenFind == lenReplace){
```

```
        for(i = 0 ; i < lenFind ; i++){
```

```
            text[index + i] = replace[i];
```

```
        }
```

```
    }
```

```
    //Replace edilecek kelime bulunan kelimedden kısa ise önce üzerine yazılır.
```

```
    else if(lenFind > lenReplace){
```

```
        for(i = 0 ; i < lenReplace ; i++){
```

```
            text[index + i] = replace[i];
```

```
        }
```

```
        //Sonrasında ise AHMET üzerine yazılan ALİ sonrası oluşan ALİET
        kelimesini
```

```
        //örnek olarak düşünersek aradaki fark 2 olduğundan İ'den sonraki her
        karakter
```

```
        //iki karakter öncesine kaydırılır.
```

```
        i = index + lenReplace;
```

```
        while(text[diff + i]){
```

```
            text[i] = text[i + diff];
```

```
            i++;
```

```
        }
```

```
        text[strlen(text) - diff] = '\0'; //String olması hasebi ile sonuna gerekli
```

```
    } //yere \0 karakteri koyulur.
```

```
    //Replace edilecek kelime bulunan kelimedden uzun ise
```

```
    else if(lenFind < lenReplace){
```

//Öncelikle sondan başlayarak bulunan kelimedden bir sonraki karakter de dahil olmak üzere

//o karaktere kadar geri geri giderek fark kadar sağa kaydırılır.

```
for(i = lenText; i >= lenFind + index ; i--){  
    text[i + lenReplace - lenFind] = text[i];  
}
```

//Sonrasında oluşan boşlukla birlikte yeterli yere replace edilecek kelime yazılır.

```
i = 0;  
int j;  
for(j = index; j < lenReplace + index ; j++) {  
    text[j] = replace[i];  
    i++;  
}  
}
```

//Search ve replace fonksiyonlarını sarmalayan fonksiyondur.

void find_and_replace(char *text, int *found_replaced){

//Gerekli bilgiler kullanıcıdan alınır.

```
printf("\n-----\n");
```

```
char find[30], replace[30], caseSensitive[4];
```

```
printf("Find: ");
```

```
scanf(" %[^\\n]s",find); //Boşlukla birlikte alması için bu şekilde bir scanf
```

yapılır.

```
printf("Replace: ");
```

```
scanf(" %[^\\n]s",replace);
```

```
printf("Case Sensitive (yes or no):");
```

```
scanf(" %s", caseSensitive);
```

```
printf("\n-----\n");
```

```
int shift_table[256];
```

```
shiftTable(find, shift_table, caseSensitive); //Shift tablomuz oluşturulur.
```

```
int index = 0;
```

//exIndex değişkeninin kullanılma sebebi:

//Tekrar tekrar arama yapıldığından eğer bulunan ve değiştirilen değerler şu şekilde ise;

//find: BEL, replace: BELIEVE birbirinin substringi oldukları için sonsuz döngüye girip

//bir önceki tur bulduğu indisi tekrar tekrar bulmasın diye bir önceki tur bulduğu indisi

//bu değişkende tutar.

```
int exIndex = -1;
```

while(index != -1){ **//Tekrar bulunamayana kadar aramaya ve değiştirmeye devam edilir.**

```

    index = search(text, find, shift_table, exIndex, caseSensitive);
    exIndex = index;
    //Buldukça değiştirilir ve değiştirilen kelime sayısını tutan değişken artırılır.
    if(index != -1){
        replace_word(text, find, replace, index);
        (*found_replaced)++;
    }
}

int main(){

    int found_replaced = 0;
    char text[MAX];

    printf("-----\n");
    read_txt(text);
    //Zamanı ölçme amacı ile asıl işi yapan fonksiyonumuzun başına ve sonuna
    zaman değişkeni atarız.
    clock_t begin = clock();
    find_and_replace(text, &found_replaced);
    clock_t end = clock();
    //Kullanıcı bildilendirmesinin yapıldığı kısım.
    printf("Process executed in %lf seconds or %lf miliseconds.\n", (double)(end-
begin)/CLOCKS_PER_SEC, (double)(end-begin)/1000);
    if(found_replaced != 0){
        printf("%d words were found and replaced.", found_replaced);
    }
    else{
        printf("Word not found!");
    }
    printf("\n-----\n");
    //Son olarak değişmiş text dizimiz dosyaya yazılır.
    write_txt(text);

    return 0;
}

```