VERİ YAPILARI VE ALGORİTMALAR

- ÖDEV 3 -

GRUP 1

İrem ATILGAN

17061036

24.04.2020

GİRİŞ

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>

FILE* openFile(char[]);
void scanFile(FILE*,char[]);
void findwBMH(char[],char[],char[],int,int*);
int replace(char[],char[],int,int);
float timedifference_msec(struct timeval,struct timeval);
```

Verilen problemde main haricinde yukarıdaki resimde gösterilen 5 farklı fonksiyon kullanılmıştır. Bunları kısaca açıklamak gerekirse:

1. **openFile** : Dosya açma işlemini gerçekleştirir.
2. **scanFile** : Dosyadaki metinin char dizisine okunması işlemini gerçekleştirir.
3. **findwBMH** : Kullanıcının girdiği kelimenin metin içinde bulunmasını sağlayan fonksiyondur. (Boyer-Moore Horspool arama algoritmasının kullanıldığı fonksiyon)
4. **replace** : Bulunan kelimenin yerinin değiştirilmesi işlemini yapar.
5. **timedifference_msec** : Ödevde istenen işlemlerin yapılma süresinin bulunmasını sağlar.

1. main

```c
int main()
{
    char word[20];
    char newWord[20];
    char fname[20];
    char c;
    int i = 0;
    int counter = 0;//count replacements
    struct timeval t0,t1;
    float elapsed;

    char csensitive;//case sensitiveness indicator

    char *text = (char*)calloc(100,sizeof(char));



    printf("\nFind : ");
    while((word[i] = getchar()) != '\n'){
        i++;
    }
    word[i] = '\0';
    printf("Replace : ");
    i = 0;
    while((newWord[i] = getchar()) != '\n'){
        i++;
    }
```

```
41      newWord[i] = '\0';
42
43      printf("Case sensitive or not [y/n] : ");
44
45      csensitive = getchar();
46
47      FILE *fp = openFile(fname);
48      scanFile(fp,text);
49
50
51      if(strlen(text) < strlen(word)){
52          printf("Input word is larger than text!..");
53          exit(1);
54      }
55
56      if(csensitive == 'y')
57          csensitive = 1;
58      else
59          csensitive = 0;
60
61
62      printf("\nText : %s",text);
63
64      gettimeofday(&t0,0);
65
66      findwBMH(word,text,newWord,csensitive,&counter);//Find with Boyer-Moore Horspool Algorithm

68      gettimeofday(&t1,0);
69
70
71      elapsed = timedifference_msec(t0, t1);
72
73      printf("\nNew Text : %s",text);
74      printf("\n\nSearching has completed!..Total change of words = %d",counter);
75      printf("\nTotal Time : %lf ms",elapsed);
76
77      freopen(fname,"w",fp);//To write the text, we have to delete existing string from the file
78      fputs(text,fp);
79
80      fclose(fp);
81      free(text);
82
83      return 0;
84  }
```

2. openFile & scanFile

```
77  FILE* openFile(char fname[]){
78
79      FILE* fp;
80
81      printf("\nFile name (txt): "); scanf("%s",fname);
82      strcat(fname,".txt");
83
84      fp = fopen(fname,"r");
85
86      return fp;
87  }
88
89  void scanFile(FILE* fp,char text[]){
90
91      while(!feof(fp)){
92
93          //fscanf(fp,"%s",text);
94          fgets(text,sizeof(char)*100,fp);
95          //fread(text,sizeof(char),,fp);
96      }
97
98
99  }
```

3. findwBMH

```
101 void findwBMH(char word[],char text[],char newWord[],int cs,int *counter){
102
103     int length = strlen(word);//the word we will be searching for
104
105     int badmatch[94];//Badmatch table takes words and punctuation marks
106
107
108
109     int i,j = length-1;
110
111     for(i = 0; i < 94; i++){
112         badmatch[i] = length;   //If the letter is not in the word we are looking for,
113                                 //we have to shift the letters at the amount of the length of the word
114     }
115
116     int offset = 32; //Space character is the offset
117
118     for(i = 0; i < length-1;i++){
119
120         badmatch[word[i]-offset] = length-i-1;
121         if(cs == 0){                                    //If It is not case sensitive, we have to fill both of the indexes
122             if(word[i] >= 'A' && word[i] <= 'Z')            //If the letter is 'a', we have to fill the index of 'A' and 'a' otherwise
123                 badmatch[word[i]-offset+'a'-'A'] = length-i-1;
124             else if(word[i] >= 'a' && word[i] <= 'z')
125                 badmatch[word[i]-offset+'A'-'a'] = length-i-1;
126         }
127     }

130     badmatch[word[i]-offset] = length;  //Last letter of the word should be the length of the word
131     i = length-1;
132     while(j >= 0 && j < strlen(text)){
133
134
135         if(text[j] == word[i]){//If two letters are identical
136
137             j--;
138             i--;
139         }
140         else{
141
142             //If the are not identical, look if the user wants case sensitiveness and in the case they don't
143             //Compare two letters (Lower Case -> Upper Case OR Upper Case -> Lower Case)
144
145             if((cs == 0) && ((text[j] - 'A') == (word[i] - 'a') || (text[j] - 'a') == (word[i] - 'A'))){
146                 i--;
147                 j--;
148
149             }
150             else{
151                 //If the letters do not match
152
153                 i = length-1;//Take index to the end of the pattern
154                 j += badmatch[text[j]-offset];//Shift letters due to the badmatch table
155             }
156         }
```

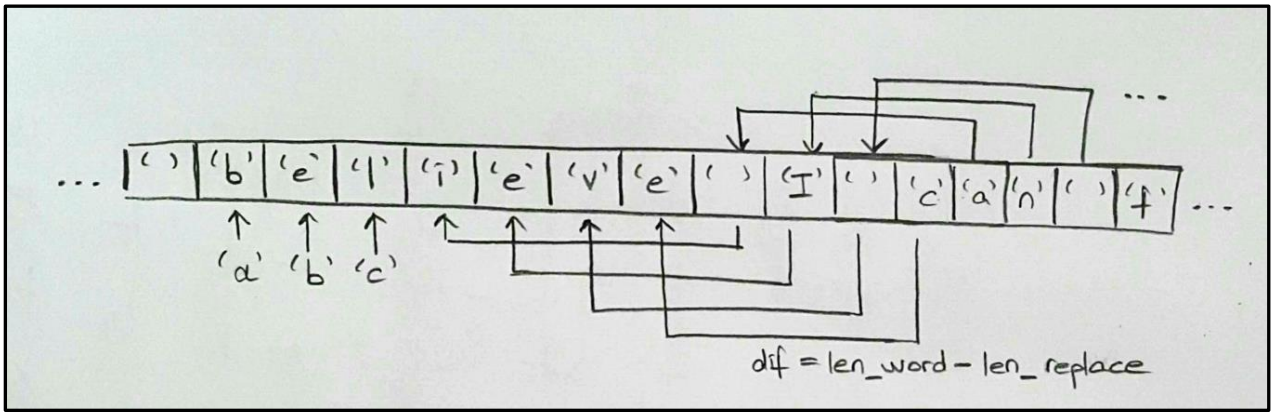
```
158         if(i < 0){//If we find the word
159 
160             struct timeval t0,t1;
161             gettimeofday(&t0,0);
162 
163             j = replace(text,newWord,j+1,length);
164 
165             gettimeofday(&t1,0);
166             float elapsed = timedifference_msec(t0,t1);
167             printf("\nTotal Time for Replacing : %lf ms",elapsed);
168 
169             (*counter)++;
170             i = length-1;   //In the case of we haven't finished the text,
171                             //We should keep looking
172 
173         }
174     }
175 
176 }
```

4. replace

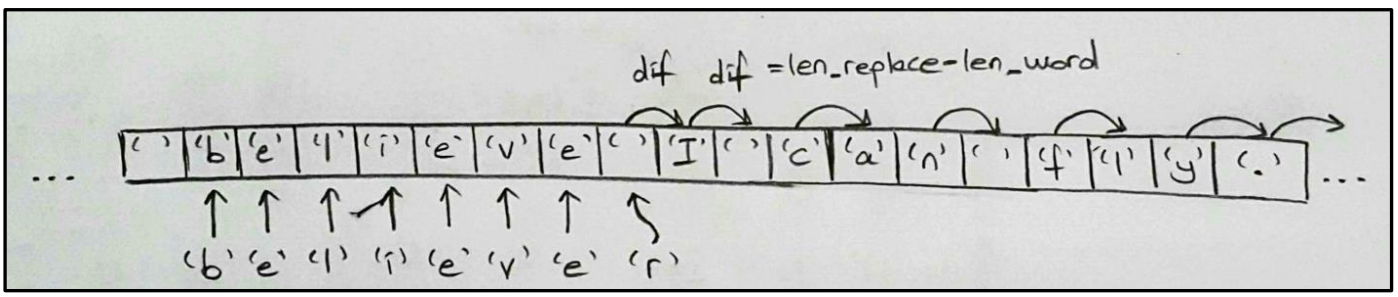
```
179 int replace(char text[], char newWord[],int text_index,int len_word){
180 
181 
182 
183     int len_text = strlen(text);
184     int len_replace = strlen(newWord);
185 
186     int k = 0, word_index = 0;
187     int dif = len_replace-len_word;                     //find the difference between lengths
188     int returnval = text_index + len_replace + len_word;//the index we will continue after replacement process
189 
190 
191     while(k < len_word && word_index < len_replace){    //Replace letters until we reach one of the word's end
192         text[text_index++] = newWord[word_index++];
193         k++;
194     }
195 
196     dif = abs(dif);
197 
198     if(word_index < len_replace){//Case 1: Word we will replace has longer length than the word will be replaced
199 
200         text = (char*)realloc(text,(len_text+dif+1)*sizeof(char));  //create space for the new letters
201 
202         for(k = len_text+dif-1;k >= text_index+dif; k--){
203             text[k] = text[k-dif];                                  //Shift other letters in the text to the end
204 
205         }
```

```
205         text[len_text+dif] = '\0';                              //Put '\0' at the end of the text to finish
206         for(k = 0; k < dif; k++){
207             text[text_index] = newWord[word_index];             //Add remaining letters to the end of the replaced word
208
209             text_index++;
210             word_index++;
211
212         }
213     }
214     else if(len_replace  == word_index && k != len_word){//Case 2: Word we will replace has shorter length than the word in the text
215
216         text_index += dif;
217         while(text_index < strlen(text)){               //Shift letters backwards
218
219             *(text+text_index-dif) = text[text_index];
220             text_index++;
221         }
222         text[text_index-dif] = '\0';
223     }
224
225     return returnval;   //After we make replacement, we return the index where we will keep searching from
226
227 }
```

CASE 1 (Yeni kelime değişecek kelimeden daha kısa ise) : Yeni kelime ile eski kelime arasında kaç harflik mesafe varsa yeni kelimeden sonra gelen her harf o mesafe kadar geriye kaydırılır.



CASE 2 (Yeni kelime değişecek kelimeden daha uzun ise): Yeni kelime ile eski kelime arasında kaç harflik mesafe alınır ve o mesafe kadar bellekte yer açılır. Daha sonra eski kelimeden sonra gelen her harf bulunan mesafe kadar sona kaydırılır. Tüm harfler kaydırıldıktan sonra yeni kelimede yerleştirilmemiş harfler de diziye eklenir.



5. timedifference_msec

```
70  float timedifference_msec(struct timeval t0, struct timeval t1)
71  {
72      return (t1.tv_sec - t0.tv_sec) * 1000.0f + (t1.tv_usec - t0.tv_usec) / 1000.0f;
73  }
```

EKRAN RESİMLERİ

```
Find : algorithm
Replace : method
Case sensitive or not [y/n] : n

File name (txt): deneme

Text : The Boyer-Moore Algorithm is considered the most efficient string matching algorithm.
Total Time for Replacing : 0.000000 ms
Total Time for Replacing : 0.000000 ms
New Text : The Boyer-Moore method is considered the most efficient string matching method.
Word Length : 9
Replaced Word Length : 6

Searching has completed!..Total change of words = 2
Total Time : 0.072000 ms
--------------------------------
Process exited after 30.26 seconds with return value 0
Press any key to continue . . .
```

```
Find : went to
Replace : visited
Case sensitive or not [y/n] : n

File name (txt): deneme

Text : Wayne went to Wales to watch walruses.
Total Time for Replacing : 0.000000 ms
New Text : Wayne visited Wales to watch walruses.
Word Length : 7
Replaced Word Length : 7

Searching has completed!..Total change of words = 1
Total Time : 0.000000 ms
--------------------------------
Process exited after 6.688 seconds with return value 0
Press any key to continue . . .
```

PROGRAMIN ÇALIŞMA SÜRESİNİN İNCELENMESİ

Kelimenin Harf Uzunluğu	Yeni Kelimenin Harf Uzunluğu	Metin Boyutu (Harf)	Değiştirilen Kelime Sayısı	Case Sensitivity	Upper Case	Süre
10	2	658	30	N	N	4,443 ms
26	3	792	14	N	Y	3,956 ms
3	26	119	16	N	Y	3,643 ms
2	10	418	30	N	Y	3,521 ms
10	2	658	30	N	Y	3,050 ms
3	26	493	14	N	Y	2,996 ms
7	2	256	12	N	Y	2,991 ms
3	26	493	14	N	N	2,988 ms
26	3	792	14	N	N	2,909 ms
26	3	487	16	N	N	2,599 ms
26	3	487	16	N	Y	2,381 ms
2	10	418	30	N	N	2,367 ms
2	7	196	12	N	Y	2,000 ms
2	7	196	12	N	N	1,995 ms
3	26	119	16	N	N	1,993 ms
7	2	256	12	N	N	1,039 ms

Tablo 1: Farklı metin, kelime, kelime boyutu kombinasyonlarıyla ölçülen çalışma süreleri

Tablo 1'de görüldüğü gibi programın çözüm süreleri test edilmiştir. Buradaki her ölçümde Case Sensitivity dikkate alınmamıştır çünkü alındığı tüm durumlarda çalışma süresi 0,000 ms olarak gözlemlenmiştir. Tablodan çıkarılan bazı sonuçlar şunlardır:

1. Çalışma süresi en çok kelime değiştirme sayısından etkileniyor. Kelime değişim sayısı arttıkça süre de doğru orantılı bir şekilde artıyor.
2. Yeni kelime, eski kelimeden daha uzunsa yeni alan yaratılması ve harflerin kaydırılması tam tersi bir duruma göre (yeni kelimenin eski kelimeden kısa olması) daha fazla zaman alıyor.
3. Aranan kelime eğer metindeki kelimenin zıttı büyüklük veya küçüklükteyse (örneğin metinde "hello" olarak geçen bir kelimeyi "HELLO" yazarak arıyorsak çalışma süresi giderek artıyor.
4. Metin boyunun da çalışma süresine her ne kadar etkisi olsa da sonuca büyük oranda bir katkısı olmuyor.

Bunun dışında aradığımız kelimenin ve diğer parametrelerin sabit tutulup metin boyutunun değiştiği durumlarda gözlenen çalışma sürelerinin tablosu ve grafiği aşağıdaki gibidir:

Metin Boyutu (Harf)	Çalışma Süresi (ms)
219	2.278
423	2.032
565	3.116
778	3.213
991	3.39
1817	5.02
2880	10.336
3602	17.802999
4809	19.945999