



ALGORİTMA ANALİZİ GRUP 2

3. Ödev

Basel Kelziye 20011906

18/11/2022

## A)YÖNTEM BÖLÜMÜ:

Dosyayı Kelime Kelime Okunur eğer Site adı okunmuşsa (“https://” ile başlayan) Son okunmuş site olarak kaydedilir. Bir sonraki siteyi okuyana kadar aradan geçen her kelimenin o sitede geçtiğini anladım.

Tablonun Key değerleri txt dosyasında bir sitede geçen kelimelerdir.

ÖRN:

<https://ce.yildiz.edu.tr> <- site adi

University Computers IT AI R&D Education <- her kelime ayrı bir key

Kelime Sayısı (Entry sayısı) ve Load Factora göre

$\text{Load Factor} = \frac{\text{Entry Sayısı}}{\text{Tablo Uzunluğu}}$  denklemini kullanarak ilk Tablo

uzunluğunu değerini elde ettim. Ondan O değere En yakın Asal sayıyı bulup Son Tablo Değeri Olarak Atadım.

En Yakın Asal Sayı Bulan Fonksiyon:

```
int find_nearest_prime(int x)
{
    int upper_bound = x;
    int lower_bound = x - 1;

    while (!isPrime(upper_bound) && !isPrime(lower_bound))
    {
        upper_bound++;
        lower_bound--;
    }
    if (isPrime(upper_bound))
        return upper_bound;
    else
        return lower_bound;

    return -1;
}
```

Tablo Uzunluğunu bulduktan sonra bulduktan sonra dinamik bir şekilde oluşturdum.

Tablodaki Her entry bir struct.

```
struct hash_entry
{
    char key[100]; // word
    struct node *head; // head pointer for linkedlist
    int isFull; // flag to check if entry is filled
};
```

### ÖNEMLİ:

Burda bir chaining mantığı yoktur. Linkli liste bir kelimenin geçtiği siteleri tutmaktadır. Aynı işlem string Array'i ile yapılabilir.

node struct ı :

```
struct node
{
    struct node *next;
    char related_sites[100];
};
```

Tablomuz Hazır olduktan sonra dosyayı bir kez daha gezerim ama bu sefer her kelimeyi tablodaki yerine **linear probing** mantığına göre yerleştiriyorum, yerleştirirken string değerini Horner metodu ile hesaplıyorum.

```

int get_horner_value(char *entry_string, int string_len, int table_len)
{
    int i;
    long long int sum = 0;
    for (i = 0; i < string_len; i++)
    {
        sum = sum + (pow(31, string_len - i)) * (entry_string[i] - 'a' + 1);
    }

    sum = (sum & 0x7fffffff) % table_len; // <-----

    if (sum < 0)
    {
        sum = sum + table_len;
    }
    return sum;
}

```

Elde ettiğimiz değer string in hashtable indeki index değeri(eğer boş ise) Sonra o indexten başlayarak tabloda boş bir yer ararız.

```

int hash_index = (int)get_horner_value(key, string_len, table_len);

int i = 0;

while (hash_table[hash_index].isFull && i < table_len && strcmp(hash_table[hash_index].key, key) != 0)
{
    hash_index = (hash_index + 1) % table_len;
    i++;
}

if (i == table_len)
{
    printf("ERROR! Table is FULL!");
    return 0;
}

```

Eğer iteratörümüz hash table boyutuna gelirse key i yerleştirecek yer bulamadı (Tablo Doldu) Demektir. Eğer O gözde o kelime önceden yerleşmişse:

```

else if (strcmp(hash_table[hash_index].key, key) == 0)
{
    struct node *new_node = create_node(site);

    if (hash_table[hash_index].head == NULL)
    {
        hash_table[hash_index].head = new_node;
    }
    else
    {
        new_node->next = hash_table[hash_index].head;
        hash_table[hash_index].head = new_node;
    }
}

```

Geçtiği sitelere yeni site ekle.

Eğer ilk defa yerleşiyorsa tabloya:

```

else
{
    struct hash_entry new_entry = create_entry(key);
    new_entry.isFull = 1;
    hash_table[hash_index] = new_entry;
    hash_table[hash_index].head = create_node(site);
    if (flag)
        printf("\n'%s' kelimesi %d. denemede yerleştirdi!\n", key, i + 1);
}

```

Yeni entry oluştur ve tabloya yerleştir.

## B) UYGULAMA BÖLÜMÜ

Anahtar bilgi: Bir kelimenin geçtiği siteler bir linkli listede tutuluyor.

Verilen expression ör: “AI ve IT” 3 string e ayırdı ve ortadaki string her zaman logical işlem olarak aldım. (eğer sadece bir kelime girilmişse AI gibi sorun yaratılmamaktadır.)

Ve Eğer logical işlem “ve” ise iki kelimenin tablodaki yerleri bulunur(eklemeye benzer olduğu için tekrar açıklamaya gerek duyulmamıştır).

Geçtiği siteleri tutan linkli listelere INTERSECTION işlemi gerçekleşir. Elde edilen linkli liste aradığımız sonuç u tutmaktadır.

```
else if (strcmp(searched_words[1], "ve") == 0)
{
    struct node *result_head = NULL;
    result_head = get_intersection_list(hash_table[hash_index].head, hash_table[hash_index2].head);
    if (result_head == NULL)
    {
        printf("\n%s ve %s nin birlikte gectikleri site bulunmamaktadır!", searched_words[0], searched_words[2]);
    }
    else
    {
        printf("\n%s ve %s nin birlikte gectikleri siteler:", searched_words[0], searched_words[2]);
        print_linked_list(result_head);
        free_list(result_head);
    }
}
```

get\_intersection\_list() methodu verilen iki listenin headini birleştirip sonuç listenin headini dönderir.

Eğer logical kelimemiz “veya” ise bu sefer iki linkli listenin UNION ı bulunur.

```
else if (strcmp(searched_words[1], "veya") == 0)
{
    struct node *result_head = NULL;
    result_head = get_union_list(hash_table[hash_index].head, hash_table[hash_index2].head);
    printf("%s veya %s \nbu sitelerden geciyor: ", searched_words[0], searched_words[2]);
    print_linked_list(result_head);
    free_list(result_head);
}
```

get\_union\_list() method benzer bir şekilde 2 ayrı head i verilen linkli listenin yeni bir listede birleştirip bize geri dönderiyor.

## Ayırılan Bellekleri temizleyen metodlar:

```
void free_list(struct node *head)
{
    struct node *tmp;
    while (head != NULL)
    {
        tmp = head;
        head = head->next;
        free(tmp);
    }
}

void free_allocated_memory(struct hash_entry *hash_table, int table_len)
{
    int i;
    for (i = 0; i < table_len; i++)
    {
        free_list(hash_table[i].head);
    }
}
```

## EXPRESSION Aratma:

Please Enter The expression you want to search  
News veya IT

## News veya IT i bulalım:

İlk önce girilen input u parse eden bir metod çağırılmaktadır.

```
void parse_input(char input[50], char parsed_input[3][50])
{
    int i = 0;
    char *token = strtok(input, " ");
    while (token != NULL)
    {
        strcpy(parsed_input[i], token);
        token = strtok(NULL, " ");
        i++;
    }
}
```



Metod girişı 3 boyutlu string array de saklıyor

Anlam sırasını koruyarak, Yani bizim `parsed_input[1]` deki değeriğimiz girişin logical expression ıdır.

**NOT:** kullanıcı 1 kelime girerse sadece kod a bir etki etmeyecektir diğer değerler boş string olduğu için.

**MOD1 de seçildiğinde:**

```
if (mod == 1)
{
    search_in_table(hash_table, searched_words, table_length);
}
```

Tabloda arama işlemi başlatılır.

Arama mantığı: iki kelimenin (varsa) tablodaki indexlerini bul, sonra geçtikleri siteleri tutan linkleri listeleri verilen expression a göre işlem yap

**veya: UNION LINKED LIST**

**ve: INTERSECTION LINKED LIST**

sonuç listemiz bizim expressionımız olmuş olur.

İki kelimenin hash indexlerini bulma:



```

int hash_index = get_horner_value(searched_words[0], strlen(searched_words[0]), table_len);
int i = 0;
int hash_index2 = get_horner_value(searched_words[2], strlen(searched_words[2]), table_len);

if (strcmp(searched_words[1], "ve") == 0 || strcmp(searched_words[1], "veya") == 0)
{
    while (strcmp(hash_table[hash_index2].key, searched_words[2]) != 0 && i < table_len)
    {
        i++;
        hash_index2 = (hash_index2 + 1) % table_len;
    }
    if (i == table_len)
    {
        printf("ERROR! The Expression you are looking for is not in the table!");
        exit(101);
    }
}

```

```

i = 0;
while (strcmp(hash_table[hash_index].key, searched_words[0]) != 0 && i < table_len)
{
    i++;
    hash_index = (hash_index + 1) % table_len;
}

```

Yine de iteratörümüz değeri tabloya eşit olduysa aradığımız eleman yoktur demek.

```

if (i == table_len)
{
    printf("ERROR! The Expression you are looking for is not in the table!");
    exit(101);
}

```

Tek kelime olduğu durum:

```

if (strcmp(searched_words[1], "ve") != 0 && strcmp(searched_words[1], "veya") != 0)
{
    printf("%s nin gectigi siteler: ", searched_words[0]);
    print_linked_list(hash_table[hash_index].head);
    // bir kelime ara.
}

```

“veya” olduğu durum:

```

else if (strcmp(searched_words[1], "veya") == 0)
{
    struct node *result_head = NULL;
    result_head = get_union_list(hash_table[hash_index].head, hash_table[hash_index2].head);
    printf("%s veya %s \n bu sitelerden geçiyor: ", searched_words[0], searched_words[2]);
    print_linked_list(result_head);
    free_list(result_head);
}

```

“ve” olduğu durum:

```

else if (strcmp(searched_words[1], "ve") == 0)
{
    struct node *result_head = NULL;
    result_head = get_intersection_list(hash_table[hash_index].head, hash_table[hash_index2].head);
    if (result_head == NULL)
    {
        printf("\n %s ve %s nin birlikte gectikleri site bulunmamaktadır!", searched_words[0], searched_words[2]);
    }
    else
    {
        printf("\n %s ve %s nin birlikte gectikleri siteler:", searched_words[0], searched_words[2]);
        print_linked_list(result_head);
        free_list(result_head);
    }
}

```

Döndürülen linkli listenin head I **NULL** ise aradığımız durum bulunmuyor demektir!

Program Testi:

Mod1:

```

Please Enter The expression you want to search
News veya IT

MODE?
1-Normal Mode
2-Detailed Mode
1
News veya IT
bu sitelerden geçiyor: -> https://ce.yildiz.edu.tr -> https://www.udemy.com -> https://www.coursera.org -> https://leetcode.com -> https://edition.cnn.com
-> https://www.youtube.com -> https://weather.com -> https://twitter.com

baselkelziye@Base ~/Desktop/3-1/algo_analizi/3.odev

```

Mod2:

Tablonun Uzunlugu -> 113  
'University' kelimesi 1. denemede yerlesti!  
  
'Computers' kelimesi 1. denemede yerlesti!  
  
'IT' kelimesi 1. denemede yerlesti!  
  
'AI' kelimesi 1. denemede yerlesti!  
  
'R&D' kelimesi 1. denemede yerlesti!  
  
'Education' kelimesi 1. denemede yerlesti!  
  
'News' kelimesi 1. denemede yerlesti!  
  
'Entertainment' kelimesi 1. denemede yerlesti!  
  
'SocialNetwork' kelimesi 1. denemede yerlesti!  
  
'Competition' kelimesi 1. denemede yerlesti!  
  
'Dataset' kelimesi 1. denemede yerlesti!  
  
'Cloud' kelimesi 1. denemede yerlesti!  
  
'Coding' kelimesi 1. denemede yerlesti!  
  
'Tutorials' kelimesi 1. denemede yerlesti!  
  
'E-Trade' kelimesi 1. denemede yerlesti!  
  
'Reviews' kelimesi 1. denemede yerlesti!  
  
'Movies' kelimesi 1. denemede yerlesti!

```
'Series' kelimesi 1. denemede yerlesti!

'Physics' kelimesi 1. denemede yerlesti!

'Blogs' kelimesi 1. denemede yerlesti!

'Business' kelimesi 1. denemede yerlesti!

'RealEstate' kelimesi 1. denemede yerlesti!

'Cars' kelimesi 1. denemede yerlesti!

'Motorcycles' kelimesi 2. denemede yerlesti!

0.
1.
2.
3.
4.
5.
6.
7.
8. Dataset -> https://www.kaggle.com
9.
10.
11. Tutorials -> https://www.tutorialspoint.com
12. Cloud -> https://www.kaggle.com
13.
14. AI -> https://www.coursera.org -> https://www.kaggle.com -> https://ce.yildiz.edu.tr
15.
16.
17. Competition -> https://leetcode.com -> https://www.kaggle.com
18. Business -> https://www.linkedin.com
19.

20. IT -> https://leetcode.com -> https://www.coursera.org -> https://www.udemy.com -> https://ce.yildiz.edu.tr
21. Cars -> https://www.motors.co.uk -> https://www.sahibinden.com
22.
23.
24. R&D -> https://ce.yildiz.edu.tr
25.
26.
27.
28.
29.
30.
31.
32.
33.
34.
35.
36.
37.
38.
39.
40. RealEstate -> https://www.sahibinden.com
41.
42.
43.
44.
45.
46. SocialNetwork -> https://www.linkedin.com -> https://twitter.com -> https://www.reddit.com -> https://www.instagram.com
47.
48.
49.
50.
51.
52. Reviews -> https://www.imdb.com -> https://www.rottentomatoes.com
53.
54. Physics -> https://www.udemy.com
55.
```

```
56.  
57.  
58.  
59.  
60. News -> https://twitter.com -> https://weather.com -> https://www.youtube.com -> https://edition.cnn.com  
61. Motorcycles -> https://www.sahibinden.com  
62.  
63.  
64.  
65.  
66.  
67.  
68. Computers -> https://leetcode.com -> https://www.udemy.com -> https://www.tutorialspoint.com -> https://ce.yildiz.edu.tr  
69.  
70.  
71.  
72. Series -> https://www.netflix.com -> https://www.imdb.com -> https://www.rottentomatoes.com  
73.  
74.  
75.  
76.  
77.  
78.  
79.  
80.  
81. Movies -> https://www.netflix.com -> https://www.imdb.com -> https://www.rottentomatoes.com  
82.  
83.  
84.  
85.  
86.  
87. Education -> https://www.coursera.org -> https://medium.com -> https://www.youtube.com -> https://www.udemy.com -> https://ce.yildiz.edu.tr  
88.  
89.  
90.
```

```
91.  
92. E-Trade -> https://www.hepsiburada.com -> https://www.amazon.com  
93.  
94.  
95. Blogs -> https://medium.com  
96.  
97.  
98.  
99.  
100.  
101.  
102. Entertainment -> https://www.netflix.com -> https://www.imdb.com -> https://medium.com -> https://www.reddit.com -> https://www.youtube.com -> https://www.rottentomatoes.com -> https://www.instagram.com  
103.  
104.  
105.  
106.  
107. Coding -> https://www.tutorialspoint.com  
108.  
109.  
110. University -> https://ce.yildiz.edu.tr  
111.  
112. News veya IT  
bu sitelerden geçiyor: -> https://ce.yildiz.edu.tr -> https://www.udemy.com -> https://www.coursera.org -> https://leetcode.com -> https://edition.cnn.com  
-> https://www.youtube.com -> https://weather.com -> https://twitter.com
```

## C) SONUÇ BÖLÜMÜ:

Load Factor = 0.1 için:

Tablonun boyutu yüksek bir miktarda arttı,

```
Tablonun Uzunlugu -> 557
'University' kelimesi 1. denemede yerlesti!
'Computers' kelimesi 1. denemede yerlesti!
'IT' kelimesi 1. denemede yerlesti!
'AI' kelimesi 1. denemede yerlesti!
'R&D' kelimesi 1. denemede yerlesti!
'Education' kelimesi 1. denemede yerlesti!
'News' kelimesi 1. denemede yerlesti!
'Entertainment' kelimesi 1. denemede yerlesti!
'SocialNetwork' kelimesi 1. denemede yerlesti!
'Competition' kelimesi 1. denemede yerlesti!
'Dataset' kelimesi 1. denemede yerlesti!
'Cloud' kelimesi 1. denemede yerlesti!
'Coding' kelimesi 1. denemede yerlesti!
```

Çoğu kelimeler ilk denemede yerleşti.

Load Factor = 0.5 için:

```
Tablonun Uzunlugu -> 113
'University' kelimesi 1. denemede yerlesti!
```

Tablonun boyutu küçüldü , çakışmalar var ama yine de az.

```
'Motorcycles' kelimesi 2. denemede yerlesti!
```

Load Factor = 0.9 için:

```
'Tutorials' kelimesi 2. denemede yerlesti!  
'E-Trade' kelimesi 1. denemede yerlesti!  
'Reviews' kelimesi 1. denemede yerlesti!  
'Movies' kelimesi 2. denemede yerlesti!  
'Series' kelimesi 1. denemede yerlesti!  
'Physics' kelimesi 2. denemede yerlesti!
```

Çakışmalar arttı, Tablonun boyutu da daraldı.

### Bonus:

Load Factor = 1 için bazı kelimelerin 15. Denemede yerleştiğini görüyoruz.

```
'RealEstate' kelimesi 15. denemede yerlesti!  
'Cars' kelimesi 13. denemede yerlesti!
```



ii)

dosya satır satır sonra kelime kelime gezdiğimiz için  
'N' kelime bulunduran dosyanın karmaşıklığı  $O(N)$ .

en yakın asal sayı bulan fonksiyonun karmaşıklığı da  $O(N)$  olduğu için, karmaşıklığı değişmiyor.

iii) Aroma motorun en büyük karmaşıklığa sahip kısım bir logical expression aromatır.

ör) "AI ve IT". avg case de tabloda eleman  
bulmanın karmaşıklığı  $O(1)$  dir. bunu varsayarak hesaplırsak  
iki linkli listenin birleşimi veya kesişimin bulan fonksiyonun  
karmaşıklığı  $O(N^2)$ .

açıklama:

bir listeyi, gezerek elemanın diğer listede olup olmadığını  
kontrol ediyoruz.

linkli liste gezmek  $\rightarrow O(N)$

linkli listede eleman arama  $\rightarrow O(N)$ .

bunu her eleman için yaptığımızda  $O(N) \cdot O(N) \rightarrow O(N^2)$

iii-1)

hash table yerine dizi kullandım. ek olarak dizide  
arama yaptığım için  $O(N)$  kadar artar karmaşıklık.  
 $O(N) \cdot O(N^2) \rightarrow O(N^3)$  olurdu.

iii-2) eğer verilen kelimelerin hashleri aynı olursa bunları  
alt alta yerleştirerek zorunda olduğumuz için (linear probing)  
diziden farklı kalmıyor aynı karmaşıklığa sahip olur

$\rightarrow O(N^3)$

VIDEO LINKİ:

[https://www.youtube.com/watch?v=1nJP0IUuw\\_E](https://www.youtube.com/watch?v=1nJP0IUuw_E)

Kaynaklar:

<https://www.cs.princeton.edu/courses/archive/fall05/cos226/lectures/hash.pdf>

<https://stackoverflow.com/questions/16400886/reading-from-a-file-word-by-word>

<https://stackoverflow.com/questions/4770985/how-to-check-if-a-string-starts-with-another-string-in-c>