İleri Algoritma Analizi 2. Ödev Raporu

Yöntem:

Dokümanlar arasındaki benzerlikleri ölçmek amacıyla tasarlanan sistemde gerçekleştirilen adımlar aşağıda özetlenmiştir:

1. Dokümanlar satır satır okunarak, parçalanmış ve sözcükler elde edilmiştir. Bu adımda sözcükler elde edilirken noktalama işaretleri göz ardı edilmiştir ve tüm harfler küçük harfe çevrilmiştir.
2. Art arda gelen 4, 5 veya 8 sözcük birleştirilerek k-shingle'lar elde edilmiştir. Shingle'lar hash tablosuna yerleştirilmiştir. Oluşturulan hash tablosu bir structure dizisidir. Structure, shingle'ı tutan bir karakter dizisi ve n adet dokümanı temsil eden bir diziden oluşmaktadır. Aşağıda oluşturulan structure gösterilmiştir.

```
typedef struct hashTable {
    char *word;
    int documents[num_of_files];
} HashTable;
```

Shingle'lar hash tablosuna yerleştirilirken, çakışma durumunu ele almak için double hashing yöntemi kullanılmıştır. Shingle'ların yerleştirildiği hash tablosu Şekil 1'de gösterilmiştir.

Hash Table:

Index	Shingle	1	2	3	4	5	6	7
0	(null)	0	0	0	0	0	0	0
1	(null)	0	0	0	0	0	0	0
2	(null)	0	0	0	0	0	0	0
3	(null)	0	0	0	0	0	0	0
4	(null)	0	0	0	0	0	0	0
5	(null)	0	0	0	0	0	0	0
6	(null)	0	0	0	0	0	0	0
7	(null)	0	0	0	0	0	0	0
8	(null)	0	0	0	0	0	0	0
9	(null)	0	0	0	0	0	0	0
10	(null)	0	0	0	0	0	0	0
11	(null)	0	0	0	0	0	0	0
12	has more than 3	2	2	2	0	2	0	0
13	(null)	0	0	0	0	0	0	0
14	(null)	0	0	0	0	0	0	0
15	(null)	0	0	0	0	0	0	0
16	(null)	0	0	0	0	0	0	0
17	maintain glamorizes the lifestyle	2	2	2	0	2	0	0
18	(null)	0	0	0	0	0	0	0
19	(null)	0	0	0	0	0	0	0
20	(null)	0	0	0	0	0	0	0
21	(null)	0	0	0	0	0	0	0
22	(null)	0	0	0	0	0	0	0
23	to target users on	2	2	2	0	2	0	0
24	(null)	0	0	0	0	0	0	0
25	(null)	0	0	0	0	0	0	0
26	to you theyve got	1	1	1	0	0	0	0
27	(null)	0	0	0	0	0	0	0
28	(null)	0	0	0	0	0	0	0
29	(null)	0	0	0	0	0	0	0
30	more than 3 million	2	2	2	0	2	0	0

Şekil 1. Hash tablosu (indis, shingle, shingle'ın dokümanlarda geçme sayısı).

3. Hash tablosu oluşturulduktan sonra, toplam eşsiz(unique) shingle sayısı ve dokümanlardaki eşsiz shingle sayıları tabloya göre hesaplanmıştır.
4. Dokümanlar çiftler halinde ele alınarak, Jaccard benzerliği hesaplanmıştır. İki doküman için jaccard benzerliği Eşitlik 1'de verilmiştir.

$$\text{Jaccard benzerliği} = \frac{1,1}{1,1 + 1,0 + 1,1} \quad (1)$$

5. Dokümanların benzerliği imza matrisi yardımıyla da hesaplanmıştır. 100 tane rastgele ve farklı hash fonksiyonu üretilerek shingle tablosunun indisleri karıştırılmıştır. Bu karıştırılmış tabloya göre dokümanlarda, shingle'ın ilk defa 1 olduğu indisler tespit edilerek imza matrisi oluşturulmuştur. Oluşturulan imza matrisi Şekil 2'de gösterilmiştir.

Signature Matrix:

# of hash function	1	2	3	4	5	6	7
1	2748	2748	2748	3404	780	657	2707
2	4262	4262	4262	3478	4262	1910	1910
3	2600	2600	2600	6103	227	453	453
4	2273	2273	4533	6064	4691	4205	4205
5	2888	2888	6073	6191	209	3037	3037
6	1716	1716	2604	4413	3626	4265	4265
7	2186	2186	2186	5065	5952	3330	3330
8	2600	2600	2600	5327	2600	4134	4134
9	2170	2170	2170	2248	2183	92	92
10	5802	5802	5802	6064	1365	1103	787
11	5219	5219	5219	1410	5764	1607	1607
12	5419	5419	5770	5518	4014	5971	5971
13	3016	3016	3016	5260	3016	1413	458
14	6186	6186	6186	4065	4386	5598	5598
15	334	334	334	1260	1260	1334	2630
16	4744	4744	4744	1145	2456	163	163
17	5290	5290	5290	5728	5728	5760	5760
18	3443	3443	3443	3164	5443	222	222
19	3226	3226	3226	1333	646	3330	3330
20	78	78	78	5260	5260	6045	6045
21	5290	5290	5290	2622	210	4695	4695
22	531	531	531	1019	2428	1061	1061
23	334	334	334	3611	4014	5819	5819
24	4744	4744	4744	5968	1225	766	766
25	3505	3505	3505	2248	3797	4673	4673
26	5802	5802	5802	2622	5322	5682	5682
27	78	78	78	2540	2540	1430	1430
28	531	531	531	5672	213	1061	1061
29	1056	1056	1056	6109	4221	1274	1274
30	6100	6100	1172	940	3697	2465	2465
31	5506	5506	5506	3164	5506	2381	2381
32	879	879	879	3965	5367	6127	6127
33	2206	2206	2206	6191	767	5819	5819
34	3130	3130	3130	38	584	1274	1274
35	4434	4434	4434	4407	4407	642	642
36	1778	1778	1658	5285	3796	1274	1274
37	4088	4088	956	345	956	5482	5482
38	26	26	26	3492	5711	2713	2713
39	465	465	465	2321	2848	3658	3658
40	4434	4434	4434	1685	5194	3011	3011

Şekil 2. İmza matrisi (hash fonksiyonu, min-hashing değerleri)

6. Oluşturulan imza matrisindeki ortak min-hashing değerlerine göre dokümanların imza benzerlik tablosu oluşturulmuştur.

Uygulama:

a)

```
Total number of 4 shingles in 7 documents: 784
Number of unique shingles in documents:
1 |450
2 |559
3 |277
4 |99
5 |338
6 |203
7 |171
```

Tablo 1. Dokümanlardaki ve toplam shingle sayısı (4 shingle)

```
Total number of 5 shingles in 7 documents: 822
Number of unique shingles in documents:
1 |454
2 |587
3 |279
4 |98
5 |339
6 |202
7 |170
```

Tablo 2. Dokümanlardaki ve toplam shingle sayısı (5 shingle)

```
Total number of 8 shingles in 7 documents: 923
Number of unique shingles in documents:
1 |466
2 |657
3 |285
4 |95
5 |341
6 |199
7 |167
```

Tablo 3. Dokümanlardaki ve toplam shingle sayısı (8 shingle)

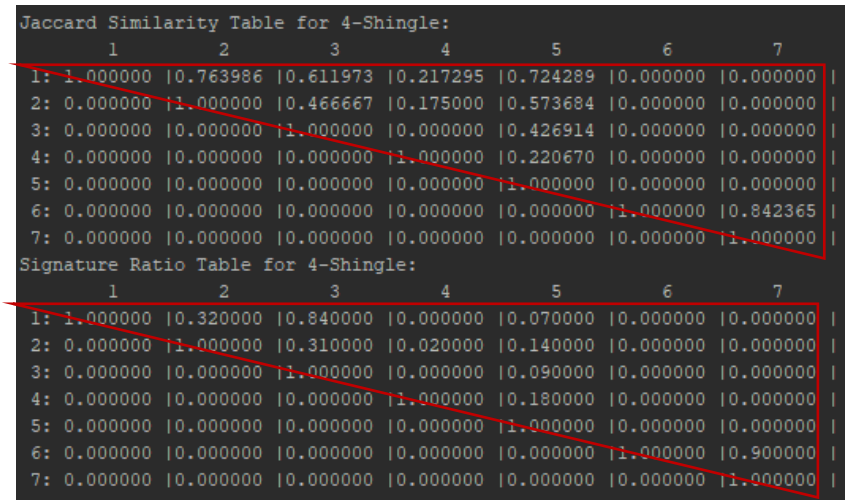
Tablo 1, Tablo 2 ve Tablo 3 incelendiğinde, shingle büyüklüğü arttığında toplam farklı shingle sayısının arttığı görülmüştür. Bunun sebebi shingle büyüklüğünün artmasıyla kelime gruplarının benzerliğinin azalması ve eşsiz(unique) shingle sayısının artmasıdır.

Dokümanlar tek tek incelendiğinde, shingle büyüklüğünün artmasıyla eşsiz shingle sayısı arasında doğrudan bir bağlantı bulunamamıştır. Dokümanın içeriğine göre eşsiz shingle sayısı artmakta ya da azalmaktadır. Ancak genel olarak shingle büyüklüğü arttıkça, eşsiz shingle sayısının arttığı söylenebilir.

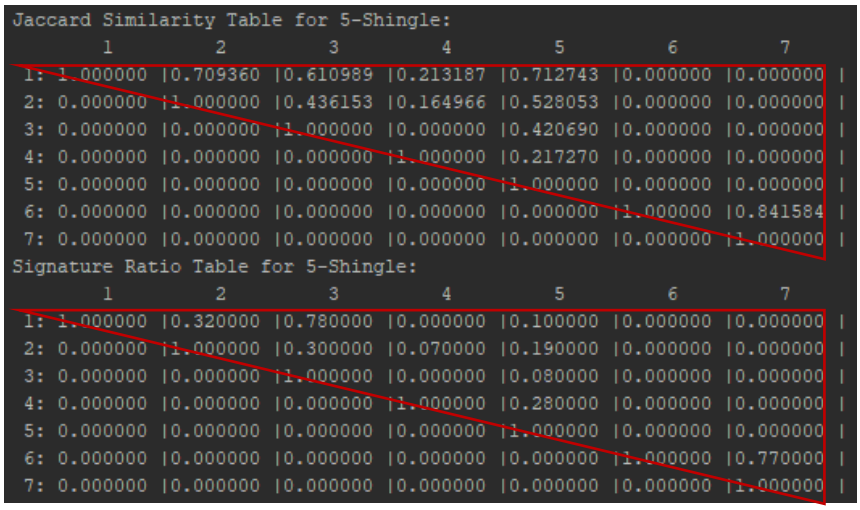
b)

Kodlanan tasarım 7 metin dosyasıyla test edilmiştir. Hazır veri seti bulunamadığı için sentetik 7 doküman oluşturulmuştur. İlk iki doküman tamamen aynı kelimleri içermesine rağmen cümlelerin yeri değiştirilerek oluşturulmuştur. 3, 4 ve 5 numaralı dokümanlar ilk dokümandan farklı miktarda parçalar alınarak oluşturulmuştur. 6 ve 7 numaralı dokümanlar ilk 5 dokümandan tamamen farklıdır. 7. doküman 6. dokümanın bir parçasıdır.

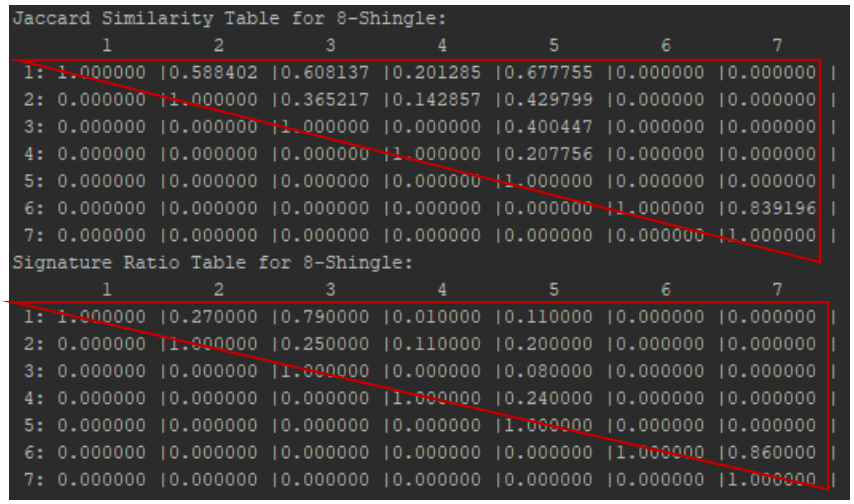
Farklı shingle büyüklüklerine göre elde edilen Jaccard benzerliği ve imza benzerliği tablosu üst diagonal matris biçiminde Tablo 4, Tablo 5 ve Tablo 6'da verilmiştir.



Tablo 4. Jaccard benzerliği ve imza benzerliği (4-shingle)



Tablo 5. Jaccard benzerliği ve imza benzerliği (5-shingle)



Tablo 6. Jaccard benzerliği ve imza benzerliği (8-shingle)

Genel olarak tablolar değerlendiriliğinde shingle büyüklüğü arttıkça doküman benzerliğinin azaldığı söylenebilir.

c)

	Jaccard Benzerliği			İmza Benzerliği		
	0.7	**0.8**	**0.9**	**0.7**	**0.8**	**0.9**
Benzer Dokümanlar →	1,2	6,7	-	1,3	1,3	6,7
	1,5				6,7	
	6,7					

Tablo 7. 4-shingle için doküman benzerliği.

7 numaralı doküman 6'nın bir parçası olduğu için imza benzerliği oldukça yüksektir. Jaccard, dokümanların benzerliğini hesaplarken, benzerleri tüme oranladığı için birbirinin parçası olan dokümanları farklı dokümanlar olarak ele alır. Halbuki imza benzerliği orana değil, shingle'ların dokümanlarda geçme durumuna baktığı için bu iki dokümanı neredeyse aynı dokümanlar olarak ele almıştır. Diğer taraftan, 1 ve 2 dokümanları aynı kelimeleri ve farklı şekilde sıralanmış cümleleri içeren dokümanlardır. Bunlar farklı dokümanlar olarak kabul edilirse, Jaccard bu dokümanları "benzer" olarak yanlış şekilde değerlendirmiştir. Ancak, imza benzerliği dokümanları "benzer olmayan" dokümanlar olarak ele almıştır.

Sonuç:

- **a)** k değeri arttıkça genelde benzerlik oranının azaldığı söylenebilir.
- **b)** k değerinin yüksek seçildiğinde benzerlikleri daha hassas biçimde hesaplandığı için, 8 değeri uygun görülmüştür.
- **c)** İmza matrisi birbirinin parçası olan dokümanları tespit etmede oldukça başarılıdır. Jaccard böyle durumlarda benzerliği yakalayamamaktadır. Bunun yanında, aynı cümleler dokümanlarda farklı sırada yer alıyorsa, Jaccard bu dokümanları "benzer" olarak ele almaktadır. Ancak imza matrisi "farklı" olarak değerlendirmektedir.

Program Kodu:

```
#include <stdio.h>
#include <math.h>
#include <malloc.h>
#include <string.h>
#include <ctype.h>
#include <windef.h>

#define num_of_files 7
typedef struct file {
    const char *key;
} File;

typedef struct hashTable {
    char *word;
    int documents[num_of_files];
} HashTable;

int compute_table_size(int word_count);
```

```
void create_hash_table(char **words, int *doc_word_count, int table_size,
int shingle_size);

int find_index(char *shingle, int table_size);

int find_second_index(char *shingle, int table_size);


char **split_line(char *line) {
//    char tokens[20][20];

    char **tokens = malloc(150 * sizeof(char *));
    int i;
    for (i = 0; i != 150; ++i) {
        tokens[i] = malloc(20 * sizeof(char));
    }

    int j, ctr;
    j = 0;
    ctr = 0;
    int flg = 0;
    char *tmp;

    if (line[0] == ' ' || line[0] == '\0' || line[0] == '\n') {
        flg = -1;
    }

    for (i = 0; i <= (strlen(line)); i++) {
        //printf("i: %d = |%c| \n", i, line[i]);
        // if space or NULL found, assign NULL into newString[ctr

        if (line[i] == ' ' || line[i] == '\0' || line[i] == '\n') {
            if (flg == 0) {
                tokens[ctr][j] = '\0';
                //printf("ctr,j: %d,%d = |%c| \n", ctr, j, '\0');
                ctr++;  //for next word
                j = 0;    //for next word, init index to 0
                flg = 1;
            }

        } else {
            if (ispunct(line[i]) == FALSE) {

//                printf("ctr,j: %d,%d = |%c| \n", ctr, j,
tolower(line[i]));
                tokens[ctr][j] = tolower(line[i]);
                j++;
                flg = 0;
            }
        }
    }

//    printf("ctr:%d\n",ctr);
    for (i = ctr; i < 150; i++)
        tokens[i] = NULL;

    return tokens;
}

void main() {
    char *files = "C:\\Users\\Mukaddes\\Desktop\\files.txt";
```

```
    FILE *fp;
    char *directory = NULL;
    size_t len = 0;
    int doc_count = 0;
    int word_count[num_of_files + 1];

    int total_word_count = 0;
    int table_size = 0;

    for (int i = 0; i < num_of_files + 1; i++)
        word_count[i] = 0;

    char **words = calloc(num_of_files * 1000, sizeof(char *));
    for (int i = 0; i != 20; ++i) {
        words[i] = calloc(20, sizeof(char));
    }

    if (words == NULL) {
        printf("Allocation Error!\n");
        exit;
    }

    fp = fopen(files, "r");
    if (fp == NULL) {
        printf("File error x!\n");
        exit;
    }

    while ((getline(&directory, &len, fp)) != -1) {
        directory[strcspn(directory, "\n")] = '\0';
//        printf("-----------------Start of doc:\n");
        FILE *fp1 = fopen(directory, "r");
        if (fp1 == NULL) {
            printf("File error y!\n");
            exit;
        }
        doc_count++;

        char *line = NULL;
        while ((getline(&line, &len, fp1)) != -1) {
//            printf("-----------------Start of line:\n");
            char **tokens = split_line(line);
            for (int i = 0; tokens[i] != NULL; i++) {

                if (total_word_count <= 1000 * num_of_files) {
//                    printf("|%s|\n", *(tokens + i));
                    words[total_word_count] = *(tokens + i);
                    word_count[doc_count]++;
                    total_word_count++;
                } else {
                    char **wordsTemp;
                    if ((wordsTemp = (char **) realloc(words, 1000 *
num_of_files)) == NULL)
                        printf("ERROR");
                    else
                        words = wordsTemp;

                    for (int k = 0; k < 1000 * num_of_files; k++) {
                        if ((wordsTemp[i] = (char *) realloc(wordsTemp[i],
20)) == NULL)
                            printf("\n\n Error...");
```

```
                        else
                            words[i] = wordsTemp[i];
                    }
//                    printf("|%s|\n", *(tokens + i));
                    words[i] = *(tokens + i);
                    word_count[doc_count]++;
                    total_word_count++;
                }
            }
        }
    }

//    for (int t = 0; t < total_word_count; t++)
//        printf("w %d: !%s!\n", t, *(words + t));

    table_size = compute_table_size(total_word_count);

    create_hash_table(words, word_count, table_size, 8);

}


void create_hash_table(char **words, int *doc_word_count, int table_size,
int shingle_size) {

    HashTable *hashtable = calloc(table_size, sizeof(HashTable));

    if (hashtable == NULL) {
        printf("Allocation Error 5!\n");
        exit;
    }

    for (int i = 0; i < table_size; i++) {
        (hashtable[i].word) = NULL;
        for (int j = 0; j < num_of_files; j++)
            hashtable[i].documents[j] = 0;
    }

    for (int i = 0; i < num_of_files + 1; i++) {
//        printf("doc count: %d\n", doc_word_count[i]);
    }
    int start = 0;
    int end = 0;
    int tmpc = 0;
    {
        for (int i = 0; i < num_of_files; i++) {
            start += doc_word_count[i];
            end = start + doc_word_count[i + 1];
            for (int j = start; j < end - (shingle_size - 1); j++) {
                char shingle[160];
                if (shingle_size == 4) {
                    snprintf(shingle, 160, "%s %s %s %s", words[j], words[j
+ 1], words[j + 2], words[j + 3]);
                } else if (shingle_size == 5) {
                    snprintf(shingle, 160, "%s %s %s %s %s", words[j],
words[j + 1], words[j + 2], words[j + 3],
                             words[j + 4]);
                } else if (shingle_size == 8) {
                    snprintf(shingle, 160, "%s %s %s %s %s %s %s %s",
words[j], words[j + 1], words[j + 2],
                             words[j + 3], words[j + 4], words[j + 5],
```

```
words[j + 6], words[j + 7]);
                }

//                printf("tmpc: %d\n", tmpc);
                //snprintf(shingle, 80, "%s %s %s %s", words[j], words[j +
1], words[j + 2], words[j + 3]);
//                printf("%d |%s\n", i, shingle);
                tmpc++;
                char *sptr = shingle;
                int h1 = find_index(shingle, table_size);
//                printf("h1: %d\n", h1);
                // place is empty, put the shingle to that index
                if (hashtable[h1].word == NULL) {
//                    printf("girdi1\n");
                    (hashtable[h1].word) = strdup(shingle);
                    hashtable[h1].documents[i] = 1;
//                    printf("|h1:%d|---|%s|\n", h1, hashtable[h1].word);
                }
                    //shingle is existing in that index, already
                else if (strcmp(shingle, hashtable[h1].word) == 0) {
//                    printf("girdi2\n");
                    hashtable[h1].documents[i]++;
//                    printf("|h1:%d|---|%s|    |doc:%d|---|num:%d|\n", h1,
shingle, i, hashtable[h1].documents[i]);
                }
                    // index of shingle is already filled
                else {
//                    printf("girdi3\n");
                    int shift = 1;
                    int h2 = find_second_index(shingle, table_size);
                    int double_ind = (h1 + shift * h2) % table_size;
                    while (hashtable[double_ind].word != NULL &&
strcmp(sptr, hashtable[double_ind].word) != 0) {
//                        printf("girdi4\n");
                        shift++;
                        if (h2 == 0)
                            h2++;
                        double_ind = (h1 + shift * h2) % table_size;

//                        printf("%d %d %d\n", h1, h2, double_ind);
                    }
                    if (hashtable[double_ind].word == NULL) {
//                        printf("girdi5\n");
                        (hashtable[double_ind].word) = strdup(shingle);
                        hashtable[double_ind].documents[i] = 1;
//                        printf("|double_ind:%d|---|%s|\n", h2, shingle);
                    } else {
//                        printf("girdi6\n");
                        hashtable[double_ind].documents[i]++;
//                        printf("|double_ind:%d|---|%s|    |doc:%d|---
|num:%d|\n", double_ind, shingle, i,
//                                hashtable[h1].documents[i]);
                    }

                }


            }

        }
```

```
    }

    int *num_of_shingles = calloc(num_of_files, sizeof(int));
    int total_shingle = 0;

    int *doc_shingles = calloc(num_of_files, sizeof(int));

    printf("Hash Table:\n");
    printf("%s%-50s", "Index|","Shingle");
    for(int i = 0; i < num_of_files; i++)
        printf("%6d",i+1);
    printf("\n");

    for (int i = 0; i < table_size; i++) {
        printf("%2d| %-50s | ", i, hashtable[i].word);
        if(hashtable[i].word != NULL)
            total_shingle++;
        for (int j = 0; j < num_of_files; j++) {
            printf("%5d ", hashtable[i].documents[j]);
            if(hashtable[i].documents[j] != 0){
                doc_shingles[j]++;
            }
            num_of_shingles[j] += hashtable[i].documents[j];
        }
        printf("\n");

    }
    printf("Total number of %d shingles in %d documents:
%d\n",shingle_size, num_of_files, total_shingle);
    printf("Number of unique shingles in documents:\n");
    for (int i = 0; i < num_of_files; i++) {
        printf("%d |%d\n", i+1, doc_shingles[i]);
    }

    double **jaccard_table = calloc(num_of_files, sizeof(double *));
    for (int i = 0; i != num_of_files; ++i) {
        jaccard_table[i] = calloc(num_of_files, sizeof(double));
    }

    if (jaccard_table == NULL) {
        printf("Allocation Error 8!\n");
        exit;
    }
 // Compute Jaccard Similarity
    for (int i = 0; i < num_of_files ; i++)
        for (int j = i; j < num_of_files; j++) {
            //printf("%d %d\n",i,j);
            int intersection = 0;
            int join = 0;
            for (int k = 0; k < table_size; k++) {
                //printf("basta %d %d\n",hashtable[k].documents[i],
hashtable[k].documents[j]);
                if (hashtable[k].documents[i] && hashtable[k].documents[j])
{
                    //printf("gırdı %d %d\n",hashtable[k].documents[i],
hashtable[k].documents[j]);
                    intersection++;
                    join++;
                } else if (hashtable[k].documents[i] ||
hashtable[k].documents[j]) {
                    //printf("GIRDI %d %d\n",hashtable[k].documents[i],
```

```
hashtable[k].documents[j]);
                    join++;
                }


            }
            double tmp = (double) intersection;
            //printf("js: %lf\n", (tmp) / join);
            jaccard_table[i][j] = ((tmp) / join);
        }

    printf("Jaccard Similarity Table for %d-Shingle:\n",shingle_size);
    for (int i = 0; i < num_of_files; i++)
        printf("%9d ",i+1);
    printf("\n");
    for (int i = 0; i < num_of_files; i++) {
        printf("%2d: ",i+1);
        for (int j = 0; j < num_of_files; j++) {
            printf("%lf |", jaccard_table[i][j]);
        }
        printf("\n");
    }
 // Create signature matrix
    int signature[100][num_of_files];

    for (int k = 0; k < 100; k++) {
        int hashfunc[table_size];
        int a = rand() % table_size;
        for (int t = 0; t < table_size; t++){
            hashfunc[t] = (a * t + 1) % table_size;
        }

        int min[num_of_files];
        for (int t = 0; t < num_of_files; t++)
            min[t] = 10000;

        for (int i = 0; i < table_size; i++) {
            //printf("i %d \n",i);
            for (int j = 0; j < num_of_files; j++) {
                //printf("i j %d %d\n",i,j);
                //printf("%d %d %d\n", hashtable[hashfunc[i]].documents[j],
min[j], hashfunc[i]);
                if (hashtable[hashfunc[i]].documents[j] == 1 && min[j] ==
10000) {
                    min[j] = hashfunc[i];
                    //printf("min %d\n",min[j]);
                }

            }
        }
        for (int t = 0; t < num_of_files; t++)
            signature[k][t] = min[t];
    }
    // Compute signature ratio
    double **signature_ratio = calloc(num_of_files, sizeof(double *));
    for (int i = 0; i != num_of_files; ++i) {
        signature_ratio[i] = calloc(num_of_files, sizeof(double));
    }

    if (signature_ratio == NULL) {
        printf("Allocation Error 8!\n");
```

```
            exit;
        }

        int intersection = 0;
        for (int i = 0; i < num_of_files ; i++) {
            for (int j = i; j < num_of_files; j++) {
                for (int k = 0; k < 100; k++) {
                    if (signature[k][i] == signature[k][j])
                        intersection++;
                }
                double tmp = (double)intersection;
                signature_ratio[i][j] = tmp / 100;
                intersection = 0;
            }
        }
        printf("Signature Ratio Table for %d-Shingle:\n",shingle_size);
        for (int i = 0; i < num_of_files; i++)
            printf("%9d ",i+1);
        printf("\n");
        for (int i = 0; i < num_of_files; i++) {
            printf("%2d: ",i+1);
            for (int j = 0; j < num_of_files; j++) {
                printf("%1f |", signature_ratio[i][j]);
            }
            printf("\n");
        }


        printf("Signature Matrix for %d-Shingle:\n",shingle_size);
        printf("%s","# of hash function|");
        for (int j = 0; j < num_of_files; j++)
            printf("%6d",j+1);
        printf("\n");
        for (int i = 0; i < 100; i++) {
            printf("%18d| ", i+1);
            for (int j = 0; j < num_of_files; j++) {
                printf("%5d ", signature[i][j]);
            }
            printf("\n");
        }
    }

    int compute_table_size(int word_count) {
        int table_size = 2 * word_count;

        int flag = 1, index, count = 0;

        while (flag == 1) {
            flag = 0;
            for (int i = 2; i <= table_size / 2; i++) {
                if (table_size % i == 0) {
                    flag = 1;
                    break;
                }
            }
            if (flag == 1)
                table_size++;
        }
    //    printf("table size:%d\n", table_size);

        return table_size;
```

```
}

int find_index(char *shingle, int table_size) {
    long horner_sum = 0;
    long power;
    int h1 = 0;

    for (int m = 0; (shingle[m] != '\0'); m++) {
        power = (long) ((pow(2, m)));
//        printf("power: %lu\n",power);
        power = power % 1021;
        horner_sum += ((int) shingle[m]) * power;
//        printf("hornersum: %lu\n",horner_sum);
//        printf("m1 = %d | pow1 = %d \n",m,horner_sum);
    }
    h1 = (int) (horner_sum % table_size);

//    printf("h1: %d| tablesize:%d\n",h1,table_size);
//    printf("horner1:%s  h1: %d \n", horner_sum, h1);

    return h1;
}

int find_second_index(char *shingle, int table_size) {
    long horner_sum = 0;
    long power;
    int h2 = 0;

    for (int m = 0; (shingle[m] != '\0'); m++) {
        power = (long) ((pow(2, m)));
        power = power % 107;
        horner_sum += ((int) shingle[m]) * power;
//        printf("m2 = %d | pow2 = %d \n",m,horner_sum);
    }
    h2 = horner_sum % 37;
//    printf("horner2:%s  h2: %d \n", horner_sum, h2);

    return h2;
}
```