BLM2512 Veri Yapıları ve Algoritmalar

Proje ödevi

Doç. Dr. Göksel Biricik

Halil İbrahim ULUOĞLU

16011093

---

1)Yöntem

Problemin Tanımı

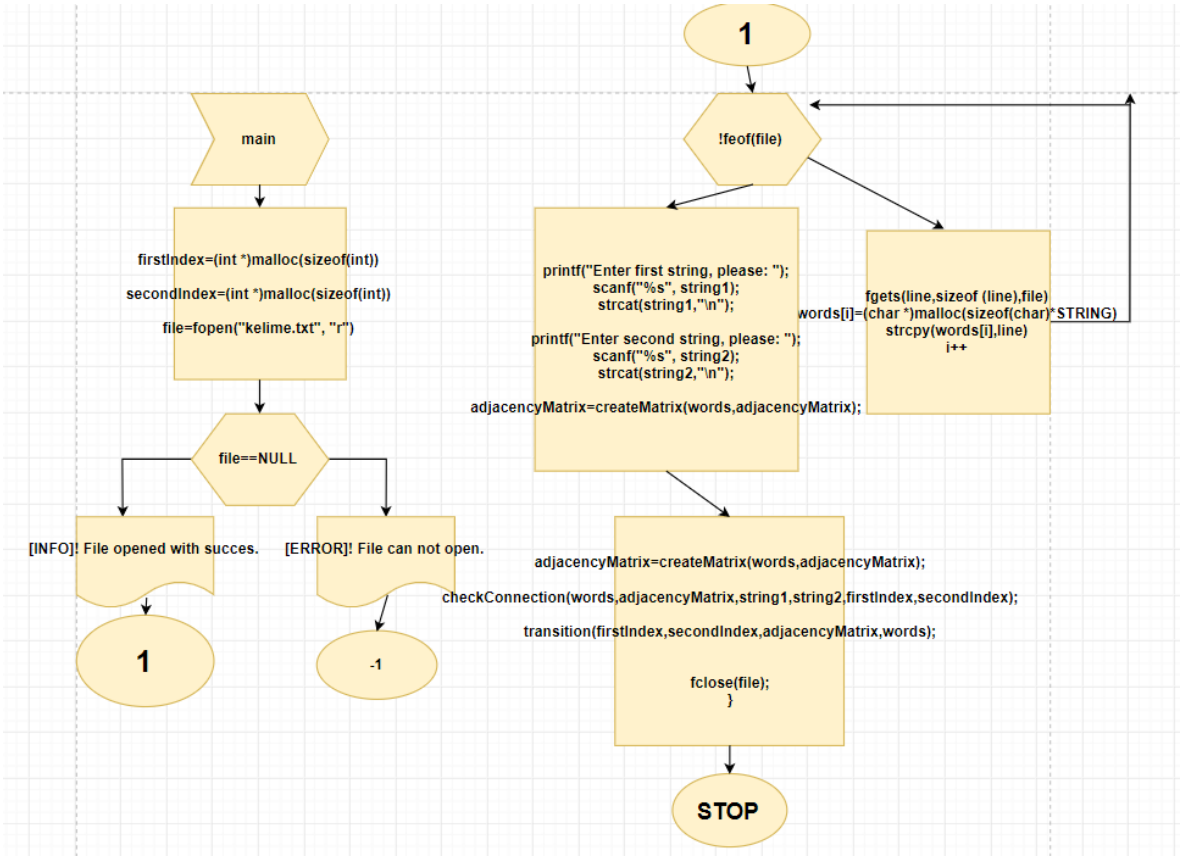
“Kelime.txt” adlı dosyada verilmiş olan her biri 5 harften oluşan kelimeler için graf yapısı oluşturulması isteniyor. Bazı kelimeler arasındaki harf değişimi 1 harf olup bu tarz kelimeler için tanım gereği komşu kelimeler deniyor. Bu kuralda olan kelimeler için komşuluk matrisinin oluşturulması isteniyor. Kullanıcının vereceği 2 kelime ile kurulmuş olan komşuluk matrisi ile çeşitli işlemler gerçekleştirerek bu 2 kelime arasındaki en kısa yolun tespit edilmesi isteniyor.

Gerçekleştirdiğim Çözüm

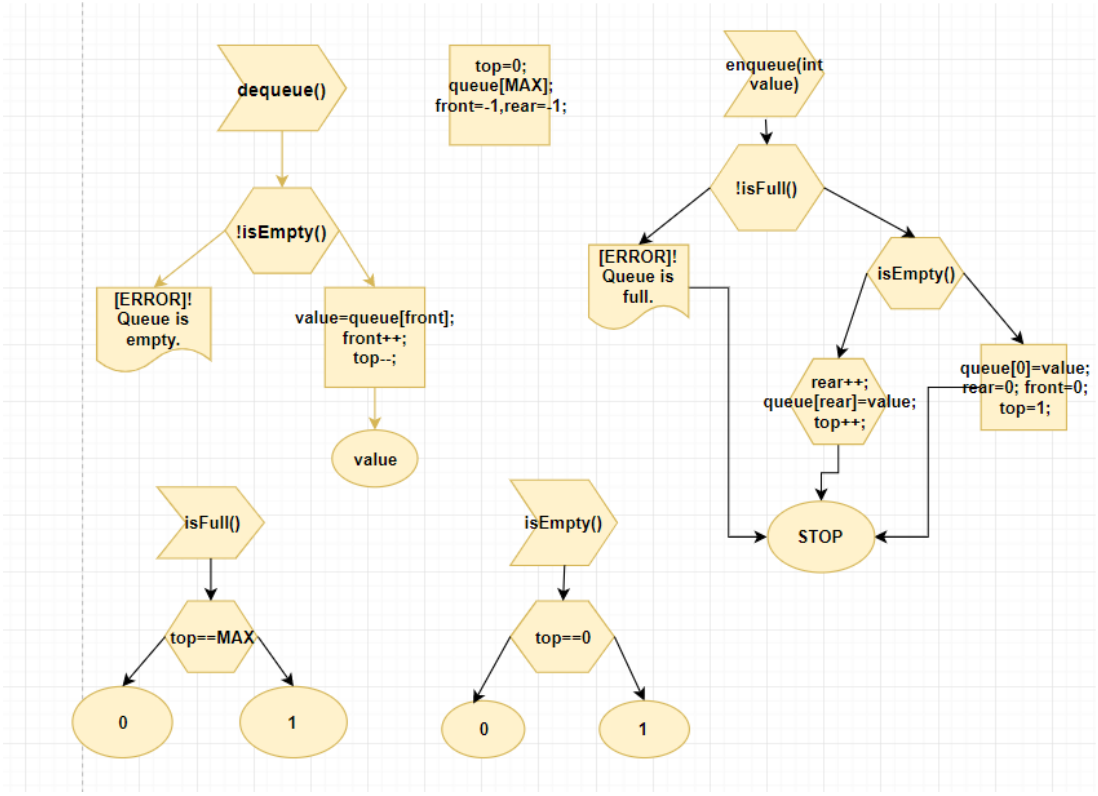
1) “Kelime.txt” adlı dosyada verilen tüm kelimeler dosyadan okuma işlemi yapılarak words adlı bir string dizisine aktarılır.
2) Harfleri arasında 1 fark bulunan kelimeleri tespit edebilmek için komşuluk matrisi oluşturulur. Bunun için verilen tüm kelimeler birbiriyle karşılaştırılıp eğer aralarında 1 harf fark varsa o matrisin ilgili satır ve sütün yerinin değeri 1 yapılır. Eğer 1 den farklı bir fark varsa 0 yapılır.
3) İlk başlangıç düğümü kuyruk yapısına ve yol dizisine konulur. Daha sonra bu düğümün var olan komşuları kuyruk yapısına konulur. Komşuları teker teker kontrol edilir. Eğer komşulardan biri istenen son kelime ise en kısa yol yazdırılır. Eğer değilse kuyrukta olup olmadığına bakılır ve yoksa onun komşuları da kuyruğa eklenir. Bu şekilde graflar üzerinden ilerleme sağlanırken ana graflar ayrı bir dizide en kısa yolun bulunabilmesi için aktarılır. Kuyrukta eleman kalmayıncaya kadar kelimeler istenen kelime ile aynı mı diye kontrol edilir.
4) İstenen şekilde ayrılan diziden en kısa ulaşılabilecek yol ekrana yazdırılır ve program sonlanır.

Akış Diyagramları

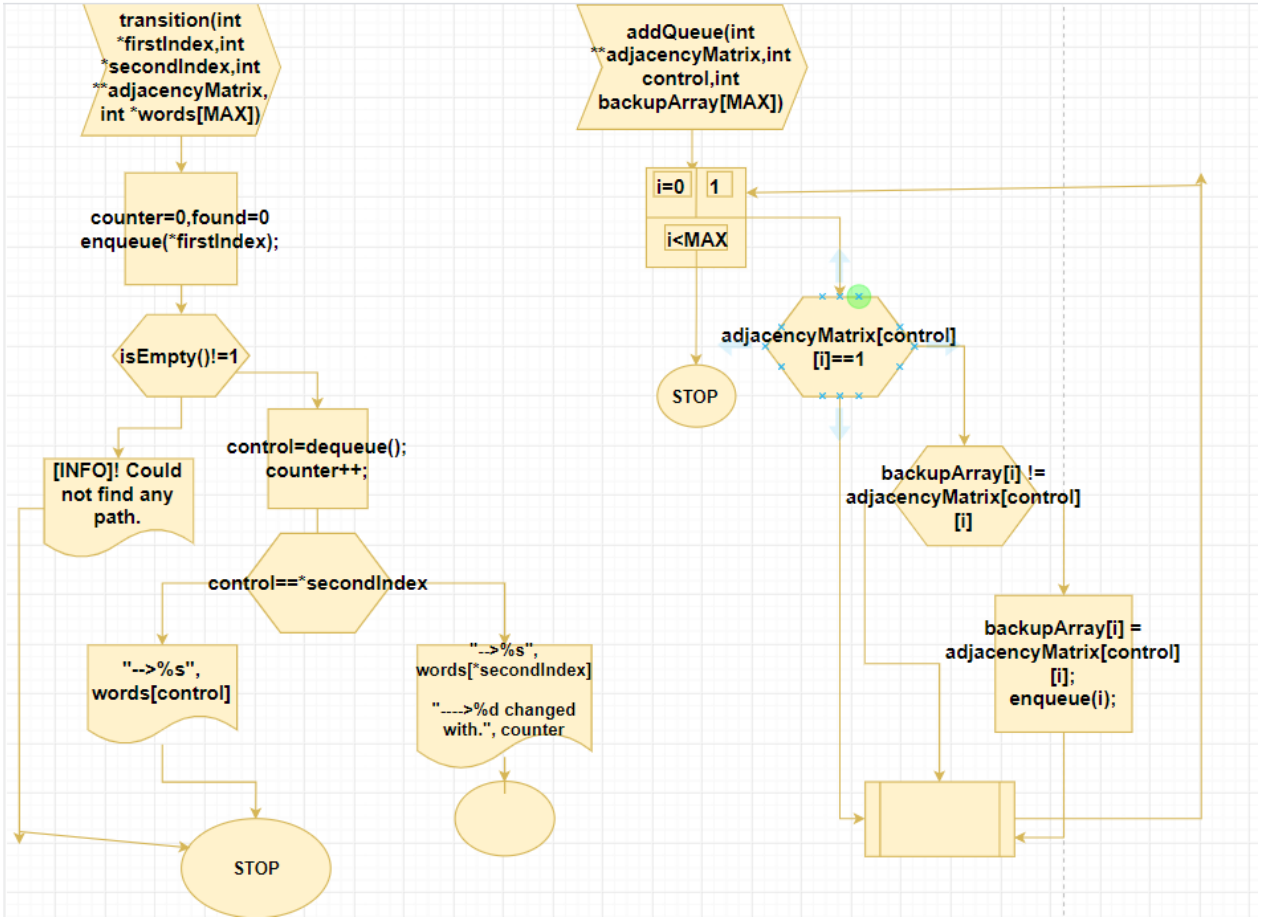
MAİN



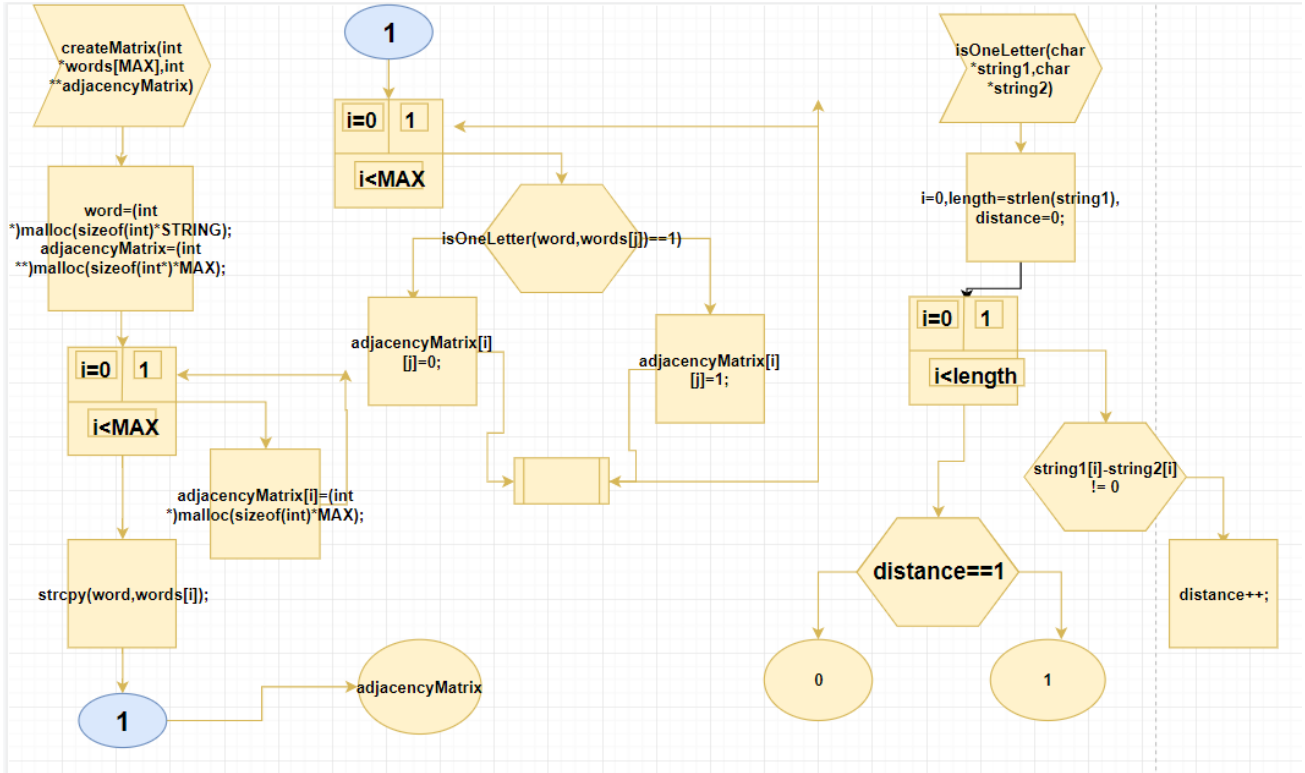
QUEUE



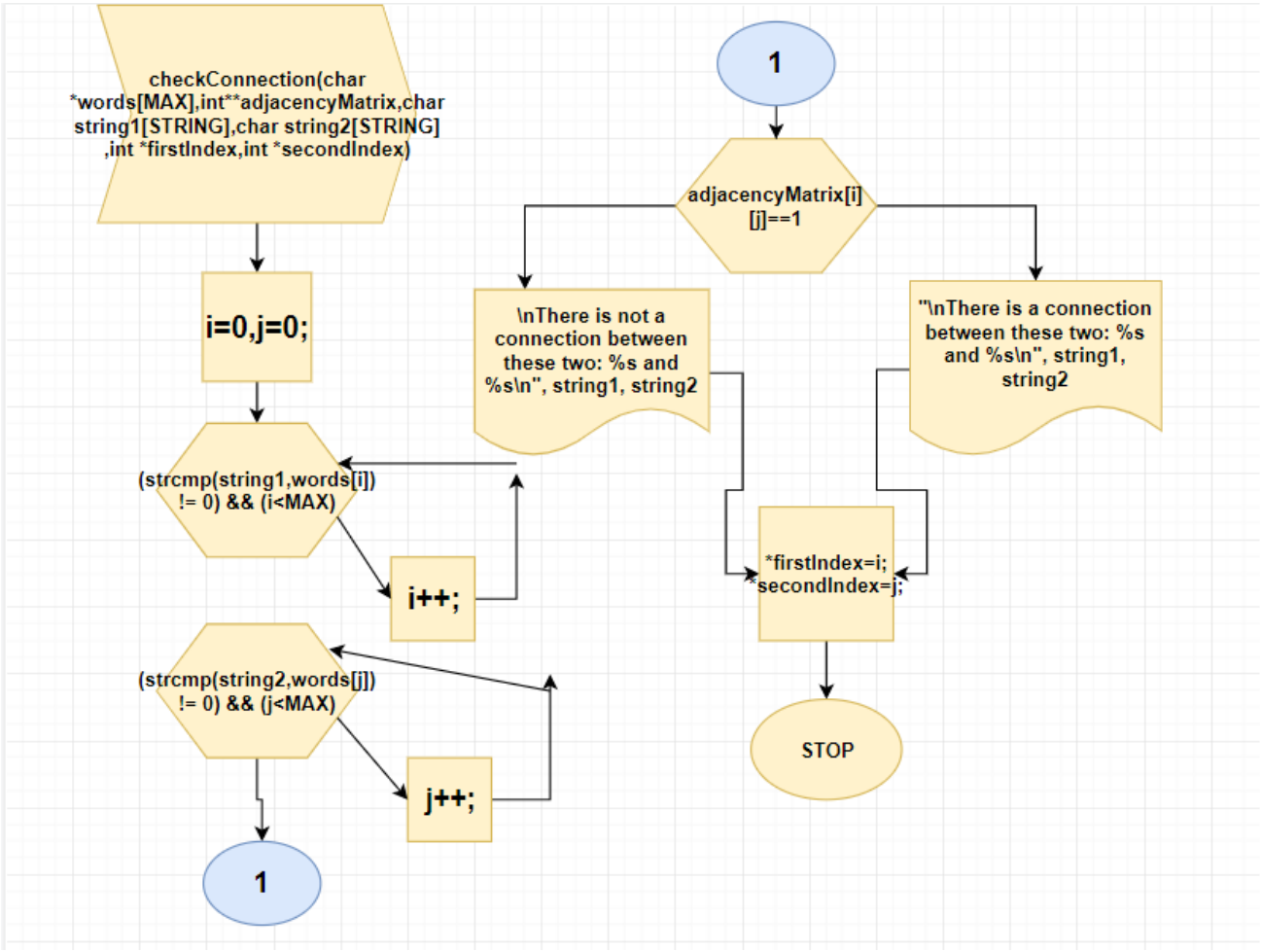
TRANSITION AND ADDQUEUE



ADJACENCYMATRIX

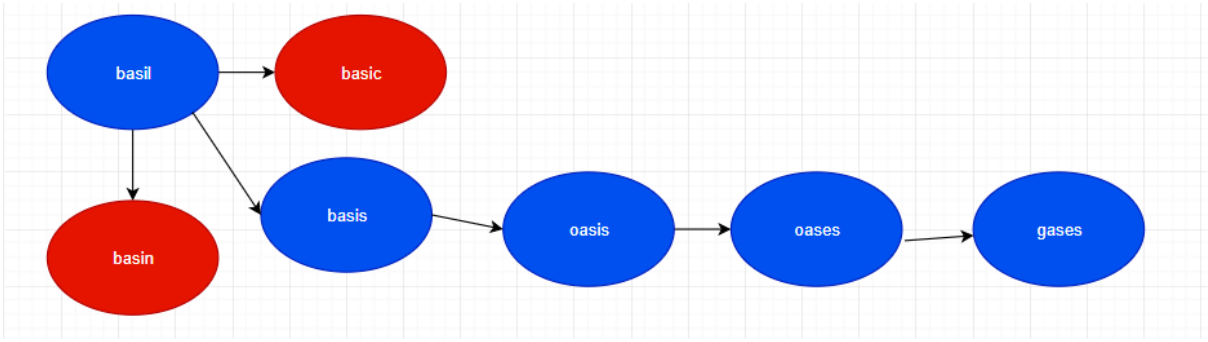


CHECKCONNECTION



2)UYGULAMA

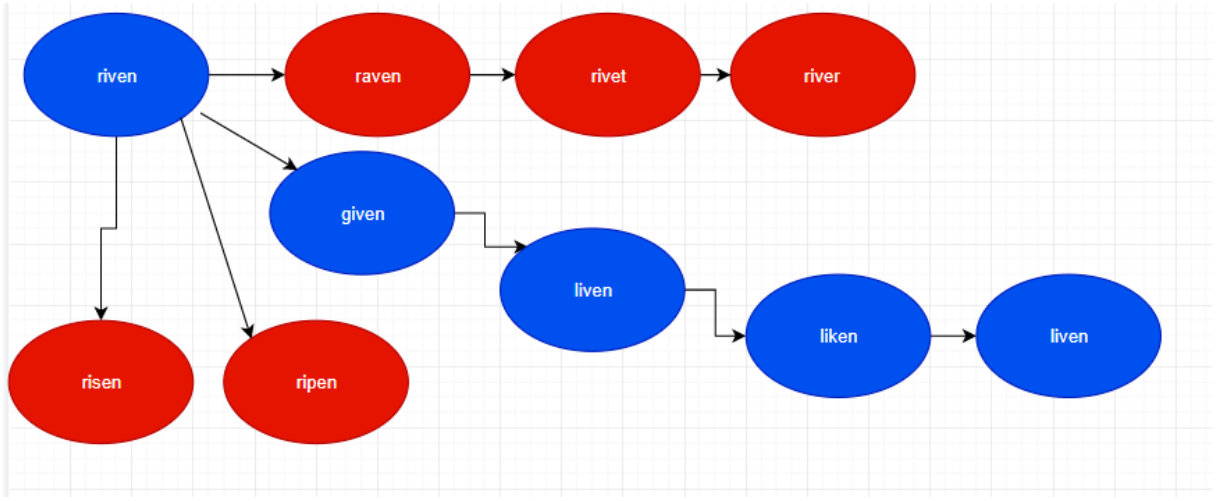
Dönüşüm Olanlar




```
C:\Users\Halil Uluoglu\Documents\2-2\Algoritma ve Veriyap²lar²\Proje\Proje.exe
[INFO]! File opened with succes.
Enter first string, please: basil
Enter second string, please: gases

There is not a connection between these two: basil
 and gases

-->basil
-->basal
-->basic
-->basin
-->basis
-->banal
-->basil
-->nasal
-->oasis
-->canal
-->natal
-->naval
-->oases
-->cabal
-->fatal
-->navel
-->gases
---->17 changed with.
```

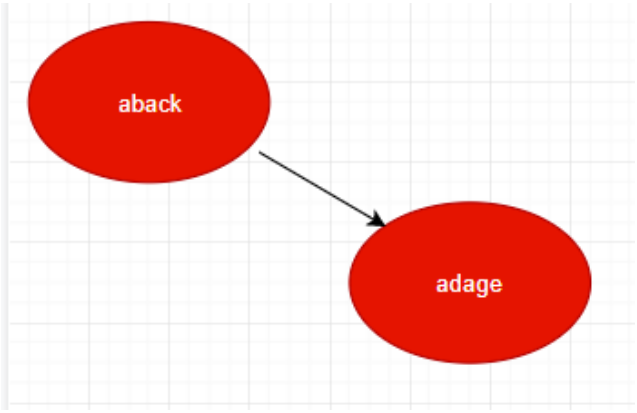


```
C:\Users\Halil Uluoglu\Documents\2-2\Algoritma ve Veriyap²lar²\Proje\Proje.exe
[INFO]! File opened with succes.
Enter first string, please: riven
Enter second string, please: linen

There is not a connection between these two: riven
 and linen

-->riven
-->given
-->liven
-->raven
-->ripen
-->risen
-->river
-->rivet
-->riven
-->liken
-->linen
---->11 changed with.
```

Dönüşüm Olmayan




```
C:\Users\Halil Uluoglu\Documents\2-2\Algoritma ve Veriyap²lar²\Proje\Proje.exe
[INFO]! File opened with succes.
Enter first string, please: aback
Enter second string, please: adage

There is not a connection between these two: aback
 and adage

-->aback
[INFO]! Could not find any path.



--------------------------------
Process exited after 2.392 seconds with return value 0
Press any key to continue . . .
```

3)KOD

```c
#include <stdio.h>
#include <stdbool.h>

#define MAX 2415                //Toplam kelime sayısı
#define STRING 10               //Bir kelime için maximum harf sayısı

int top=0;                      //Queue veriyapısı için dizi ve en üst işaretcisi
int queue[MAX];
int front=-1,rear=-1;

//Bu fonksiyon txt dosyasında bulunan her bir kelimenin birbiriyle olan harf farkını bulmakta.
//Eğer bu harf farkı 1 ise bunu adjacencyMatrix dediğim bir komşuluk matrisinde tutuyor.
int ** createMatrix(int *words[MAX],int **adjacencyMatrix)
{
    int i,j;
    char *word;

    word=(int *)malloc(sizeof(int)*STRING);
    adjacencyMatrix=(int **)malloc(sizeof(int*)*MAX);
    for(i=0; i<MAX; i++)
    {
        adjacencyMatrix[i]=(int *)malloc(sizeof(int)*MAX);
    }
    for(i=0; i<MAX; i++)
    {
        strcpy(word,words[i]);
        for(j=0; j<MAX; j++)
        {
            if(isOneLetter(word,words[j])==1)
            {

              adjacencyMatrix[i][j]=1;
            }
            else
            {
              adjacencyMatrix[i][j]=0;
            }

        }
    }
    return adjacencyMatrix;

}

//Bu fonksiyonda createMatrix fonksiyonu için gerekli olan kelimeler arasında 1 harf
//karşılaştırması yapıyor.Eğer kelimeler arasında bir 1 harf fark varsa 1 dönderiyor.
int isOneLetter(char *string1,char *string2)
{
    int i=0,length,distance=0;
    length=strlen(string1);
    for(i=0; i<length; i++)
    {
        if(string1[i]-string2[i] != 0)
        {
            distance++;
        }
    }

    if(distance==1)
    {
        return 1;
    }
    else
    {
        return 0;
    }

}

//Bu fonksiyon verilen 2 tane kelimenin komşuluk matrisindeki konumunu buluyor.
void checkConnection(char *words[MAX],int**adjacencyMatrix,char string1[STRING],char string2[STRING]
                     ,int *firstIndex,int *secondIndex)
{
    int i=0,j=0;

    while((strcmp(string1,words[i]) != 0) && (i<MAX))
    {
        i++;
    }
```

```
    while((strcmp(string1,words[i]) != 0) && (i<MAX))
    {
        i++;
    }
    while((strcmp(string2,words[j]) != 0) && (j<MAX))
    {
        j++;

    }

    if(adjacencyMatrix[i][j]==1)
    {
        printf("\nThere is a connection between these two: %s and %s\n", string1, string2);
    }
    else
    {
        printf("\nThere is not a connection between these two: %s and %s\n", string1, string2);
    }

    *firstIndex=i;
    *secondIndex=j;
}

//Tüm bfs işlemlerinin yapıldığı fonksiyon.(Gerekli açıklama raporda yapılmıştır.)
//Aynı zamanda en kısa yolun bulunduğu fonksiyon.(Gerekli açıklama raporda yapılmıştır.)
int *transition(int *firstIndex,int *secondIndex,int **adjacencyMatrix,
                int *words[MAX])
{
    int control,counter=0,found=0;
    int backupArray[MAX];

    enqueue(*firstIndex);

    while(isEmpty()!=1)
    {
        control=dequeue();
        counter++;
        if(control==*secondIndex)
        {
            printf("-->%s", words[*secondIndex]);
```

```
            counter++;
            if(control==*secondIndex)
            {
                printf("-->%s", words[*secondIndex]);
                printf("---->%d changed with.", counter);
                return;
            }
            else
            {
                printf("-->%s", words[control]);
                addQueue(adjacencyMatrix,control,backupArray);
            }
        }
        printf("[INFO]! Could not find any path.\n");
}

//Komşuluk matrisinde bulunan elemanların kuyruğa eklenmesi.
void addQueue(int **adjacencyMatrix,int control,int backupArray[MAX])
{
    int i;
    for(i=0; i<MAX; i++)
    {
        if(adjacencyMatrix[control][i]==1)
        {
            if(backupArray[i] != adjacencyMatrix[control][i])
            {
                backupArray[i] = adjacencyMatrix[control][i];
                enqueue(i);
            }
        }
    }
}

//isEmpty()-->kuyruğun boş olup olmadığını kontrol ediyor.
//isFull()-->kuyruğun dolu olup olmadığını kontrol ediyor.
//enqueue()-->Kuyruğa eleman eklemek için kullanılan fonksiyon.
//dequeue()-->Kuyruğa eleman çıkarmak için kullanılan fonksiyon.
int isEmpty()
{
    if(top==0)
```

```
145  //isEmpty()-->kuyruğun boş olup olmadığını kontrol ediyor.
146  //isFull()-->kuyruğun dolu olup olmadığını kontrol ediyor.
147  //enqueue()-->Kuyruğa eleman eklemek için kullanılan fonksiyon.
148  //dequeue()-->Kuyruğa eleman çıkarmak için kullanılan fonksiyon.
149  int isEmpty()
150  {
151      if(top==0)
152      {
153          return 1;
154      }
155      else
156      {
157          return 0;
158      }
159  }
160
161  int isFull()
162  {
163      if(top==MAX)
164      {
165          return 1;
166      }
167      else
168      {
169          return 0;
170      }
171  }
172
173  void enqueue(int value)
174  {
175      if(!isFull())
176      {
177          if(isEmpty())
178          {
179              queue[0]=value;
180              rear=0; front=0;
181              top=1;
182          }
183          else
184          {
```

```
181              top=1;
182          }
183          else
184          {
185              rear++;
186              queue[rear]=value;
187              top++;
188          }
189      }
190      else
191      {
192          printf("[ERROR]! Queue is full.\n");
193      }
194  }
195  int dequeue()
196  {
197      int value;
198      if(!isEmpty())
199      {
200          value=queue[front];
201          front++;
202          top--;
203          return value;
204      }
205      else
206      {
207          printf("[ERROR]! Queue is empty.\n");
208      }
209  }
210
211  //Gerekli tüm veriyapılarının oluşturulduğu,2 tane kelimenin kullanıcıdan alındığı
212  //dosyanın okunma işlemi ve gerekli fonksiyonların çağırıldığı main fonksiyonu.
213  void main()
214  {
215      FILE *file;
216      char *words[MAX];
217      char line[STRING],temp[STRING],string1[STRING],string2[STRING];
218      int i=0,j=0,k;
219      int **adjacencyMatrix;
220      int *firstIndex,*secondIndex;
```

```
    }
}

//Gerekli tüm veriyapılarının oluşturulduğu,2 tane kelimenin kullanıcıdan alındığı
//dosyanın okunma işlemi ve gerekli fonksiyonların çağırıldığı main fonksiyonu.
void main()
{
    FILE *file;
    char *words[MAX];
    char line[STRING],temp[STRING],string1[STRING],string2[STRING];
    int i=0,j=0,k;
    int **adjacencyMatrix;
    int *firstIndex,*secondIndex;
    int backupArray[MAX];

    firstIndex=(int *)malloc(sizeof(int));
    secondIndex=(int *)malloc(sizeof(int));

    file=fopen("kelime.txt", "r");
    if(file==NULL)
    {
        printf("[ERROR]! File can not open.\n");
        return -1;
    }
    else
    {
        printf("[INFO]! File opened with succes.\n");
    }

    while(!feof(file))
    {

        fgets(line,sizeof (line),file);
        words[i]=(char *)malloc(sizeof(char)*STRING);
        strcpy(words[i],line);
        i++;
    }

    printf("Enter first string, please: ");
    scanf("%s", string1);
    strcat(string1,"\n");

    printf("Enter second string, please: ");
    scanf("%s", string2);
    strcat(string2,"\n");

    adjacencyMatrix=createMatrix(words,adjacencyMatrix);

    checkConnection(words,adjacencyMatrix,string1,string2,firstIndex,secondIndex);

    transition(firstIndex,secondIndex,adjacencyMatrix,words);


    fclose(file);
}
```