T.C.
YILDIZ TEKNİK ÜNİVERSİTESİ
ELEKTRİK-ELEKTRONİK FAKÜLTESİ
BİLGİSAYAR MÜHENDİSLİĞİ

KONU: QUEUE IMPLEMENTATION ON GRAPH DATA STRUCTURE

VERİ YAPILARI VE ALGORİTMALAR

3.ÖDEV

BAHAR-(2016-2017)

Ders Adı:
VERİ YAPILARI VE ALGORİTMALAR

Ders Öğretmeni:
Doç. Dr. Mine Elif KARSLIGİL

Hazırlayan:
Adı : MUHAMMED YASİN
Soyadı : SAĞLAM
Numarası: 15011804

TARİH (05.05.2017)

1.YÖNTEM

Soruda öncelikle problemin çözümünde kelimelerin bulunduğu kelime dosyasından kelimelerin çekilerek iki kelime arasındaki tek harf değişimine bakılarak bir graph veya komşuluk matrisi yapısı oluşturulup kelimelerin oluşturulan bu yapıya komşuluk durumlarına uygun olarak yerleştirilmesi istenmektedir. Sonrasında ise kuyruk yapısı kullanılarak girilen iki kelime arasında dönüşüm varsa kaç adımda olduğu ve dönüşüm aşamasında kullanılan kelimelerin tamamının kullanıcıya gösterilmesi istenmektedir.

Problemin çözümünde indisler üzerinden işlem yapılarak arama ve string kıyaslama işlemlerinin optimize edilmesine ek olarak, yer ve zaman bakımından program optimizasyonu boşluk içermeyen bir komşuluk yapısı ve kuyruk yapısı tanımlanmasıyla ve modüllerin bu yapılara uygun dizayn edilmesiyle sağlanmıştır.

TANIMLANAN YAPILAR;

1) **Optimizasyona uyumlu komşuluk matrisi**

```
typedef struct {
    int *neighbours; //komşuların indislerinin tutulduğu dizi
    int neighbour_count; //komşu sayısı
}ADJACENCY;
```

2) **Kuyruk yapısı**

```
struct node{
  int info; //kuyruk node una  gonderilecek indis degerini tutacak olan degisken
  struct node *ptr; //bir sonraki kuyruk node unun adresini tutan degisken
}*front,*rear,*temp,*front1;
```

KULLANILAN YAPISAL FONKSİYONLAR;

1) İki kelime arasında tek harf değişimi var ise 1 yok ise 0 döndüren **int one_difference(char *word1,char *word2)** prototipli fonksiyon.

2) Kuyruk fonksiyonları;

int front_element(); //kuyruğun ilk elemanını dışarı döndüren fonksiyon

void create(); //kuyruk oluşturan fonksiyon

void dequeue(); //kuyruktan eleman silen fonksiyon

void enqueue(int value); //kuyruğa eleman ekleyen fonksiyon

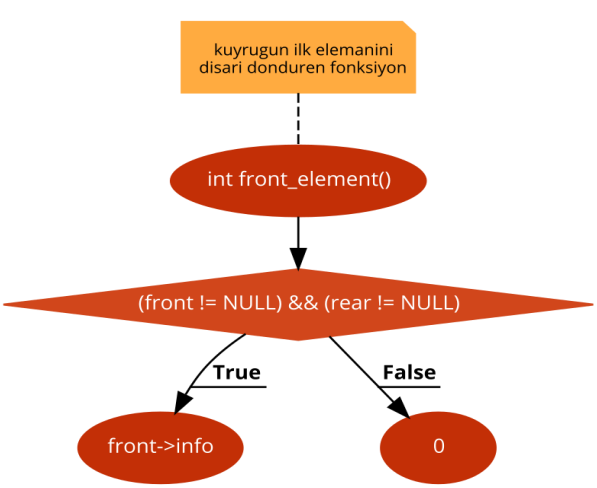
int isEmpty(); //kuyruk boş mu değil mi kontrol eden fonksiyon

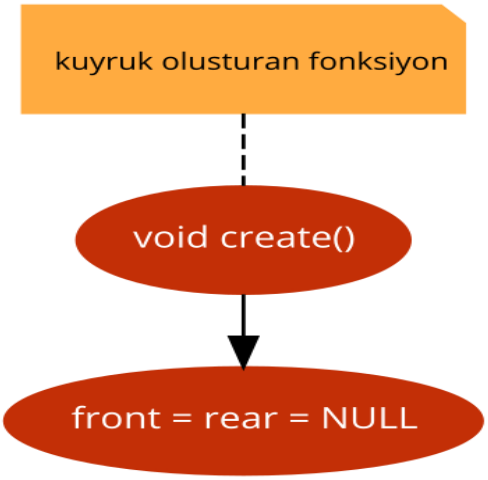
3) Komşuluk matrisinin ilk değer ataması işlemlerini yapan fonksiyon

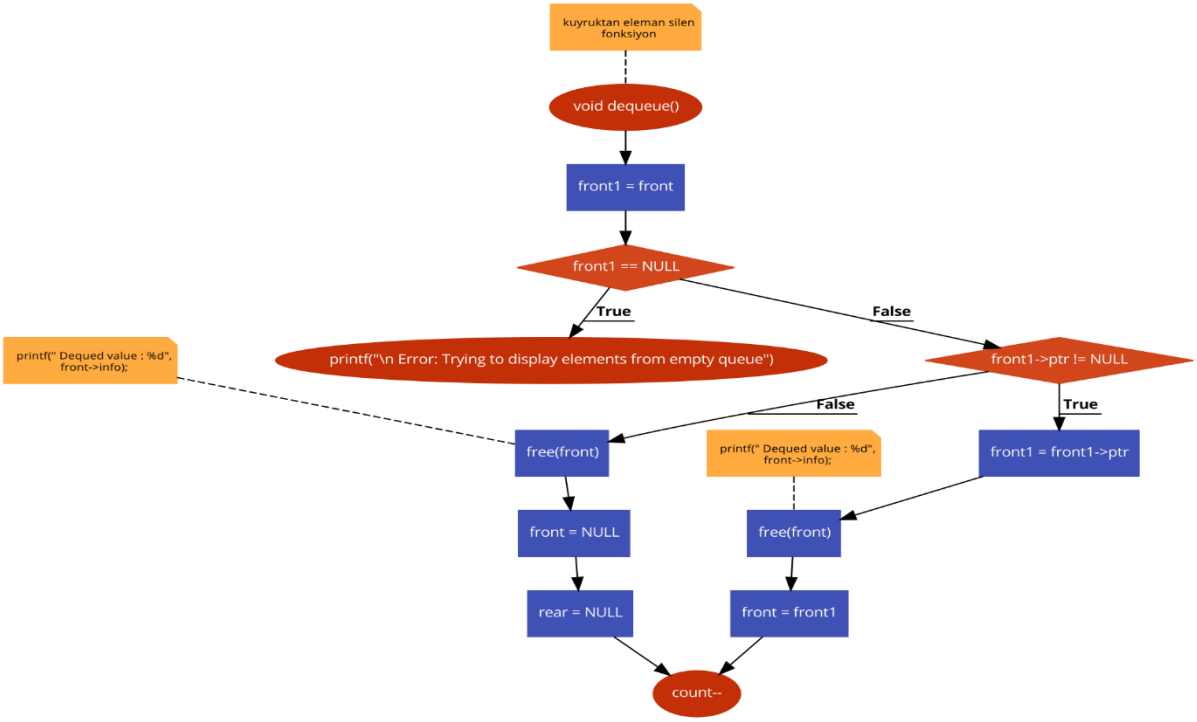
void initialize_adjacency(ADJACENCY **words,int size);

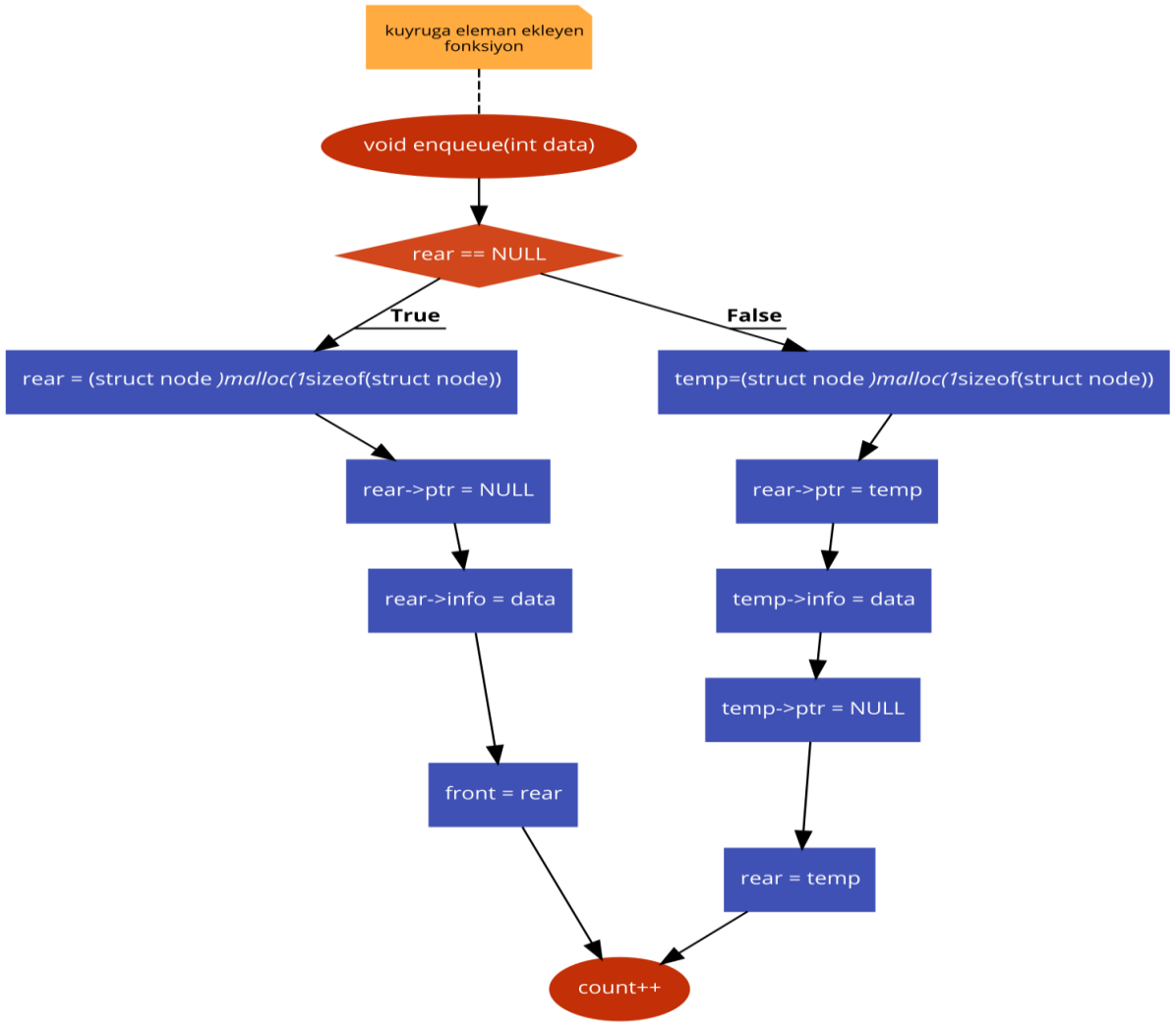
Çözümü olarak, tanımlanan yapılar ve oluşturulan fonksiyonlar yardımıyla öncelikle kelimeler dosyasındaki herbir kelime dinamik olarak oluşturulan **list[]** dizisinde tutulmuştur. (Her yapılan kelime okuma işleminde bir eleman eklenmiştir.) Toplam kelime sayısı ise satır sayısına eşittir ve **line** adı verilen değişkende saklanmıştır. Okunan kelimeler **ADJACENCY tipinde oluşturulan words** yapısına her bir kelimenin komşulukları **one_difference fonksiyonu** ile bulunarak komşuluk indisleri **words** yapısı içerisindeki **neighbours** dizisine dinamik olarak yerleştirilmiştir. Böylelikle boşluk içermeyen bir komşuluk matrisi elde edilmiştir. Sonrasında kullanıcıdan kelime listesi içerisindeki iki kelime alınmıştır ve indisleri listeden bulunarak **index1** ve **index2** değişkenlerine atanmıştır. **index1** değeri(kaynak olarak alınan ilk kelimenin indisi) oluşturulan dinamik kuyruk yapısına eklenmiştir, eklenen bu değer kuyruktan çekilerek **index2** değerine(hedef olarak alınan ikinci kelimenin indisi) eşitliği kontrol edilerek dönüşüm durumu kontrol edilmiştir. Eşitlik varsa döngüden çıkılmıştır, yoksa çekilen indisin komşu indisleri oluşturulan dinamik yapıdaki yedek dizi yardımıyla kontrol edilerek (aynı düğümün tekrardan kuyruğa eklenmemesi için) kuyruğa ve yedek diziye eklenmiştir. İterasyonlar kuyrukta hiç eleman kalmayana kadar(kelimeler arasında dönüşüm olmadığını ifade etmektedir) veya kuyruktan çekilen indisin **index2** ye eşit olması durumuna kadar (dönüşümün gerçekleşmesi durumu **found** değişkeni kullanılarak kontrol edilmiştir) iteratif olarak devam ettirilmiştir. Eğer dönüşüm var ise kaç adımda ve hangi kelimeler yoluyla yapıldığının bulunması için kuyruğa ekleme yapılırken kıyaslama yapılan **tmp_arr[]** dizisi yardımıyla optimize bir şekilde ters mühendislik yapılarak (childden parenta doğru bakılarak yani iteratif olarak dizinin en başından en sonuna doğru son indisin komşusu var mı kontrolü yapılarak) ilgili indislerdeki kelimeler ve dönüşümün kaç komşudan geçilerek gerçekleştiği(yapılan işlem sayısı) ekrana yazdırılmıştır.

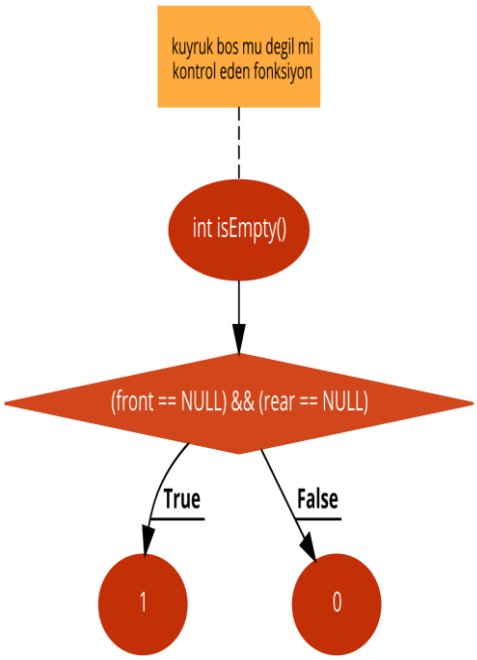
AKIŞ DİYAGRAMLARI

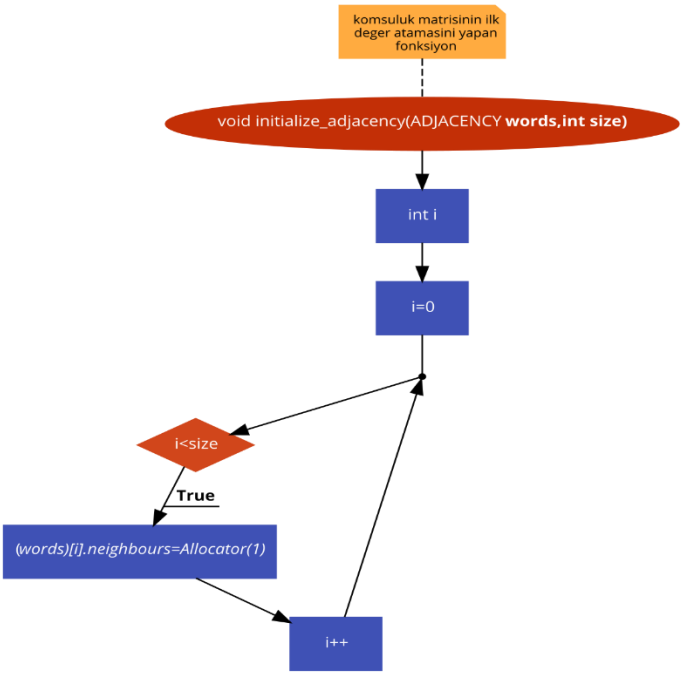


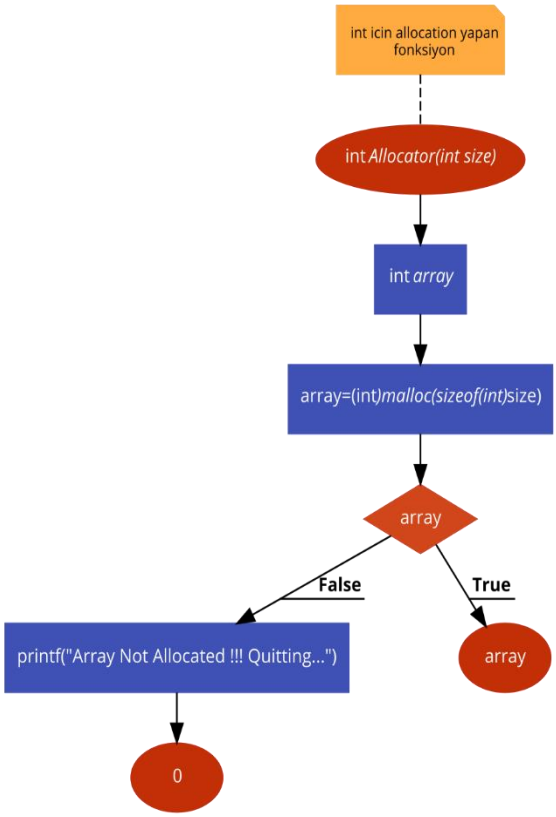


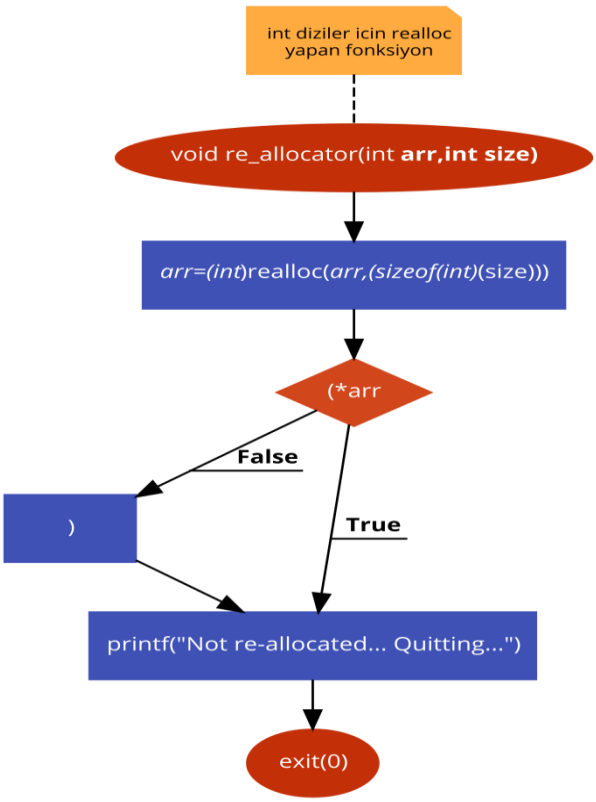

kuyruktan eleman silen fonksiyon
void dequeue()
front1 = front
front1 == NULL
True
False
printf(" Dequed value : %d", front->info);
printf("\n Error: Trying to display elements from empty queue")
front1->ptr != NULL
False
True
free(front)
printf(" Dequed value : %d", front->info);
front1 = front1->ptr
front = NULL
free(front)
rear = NULL
front = front1
count--

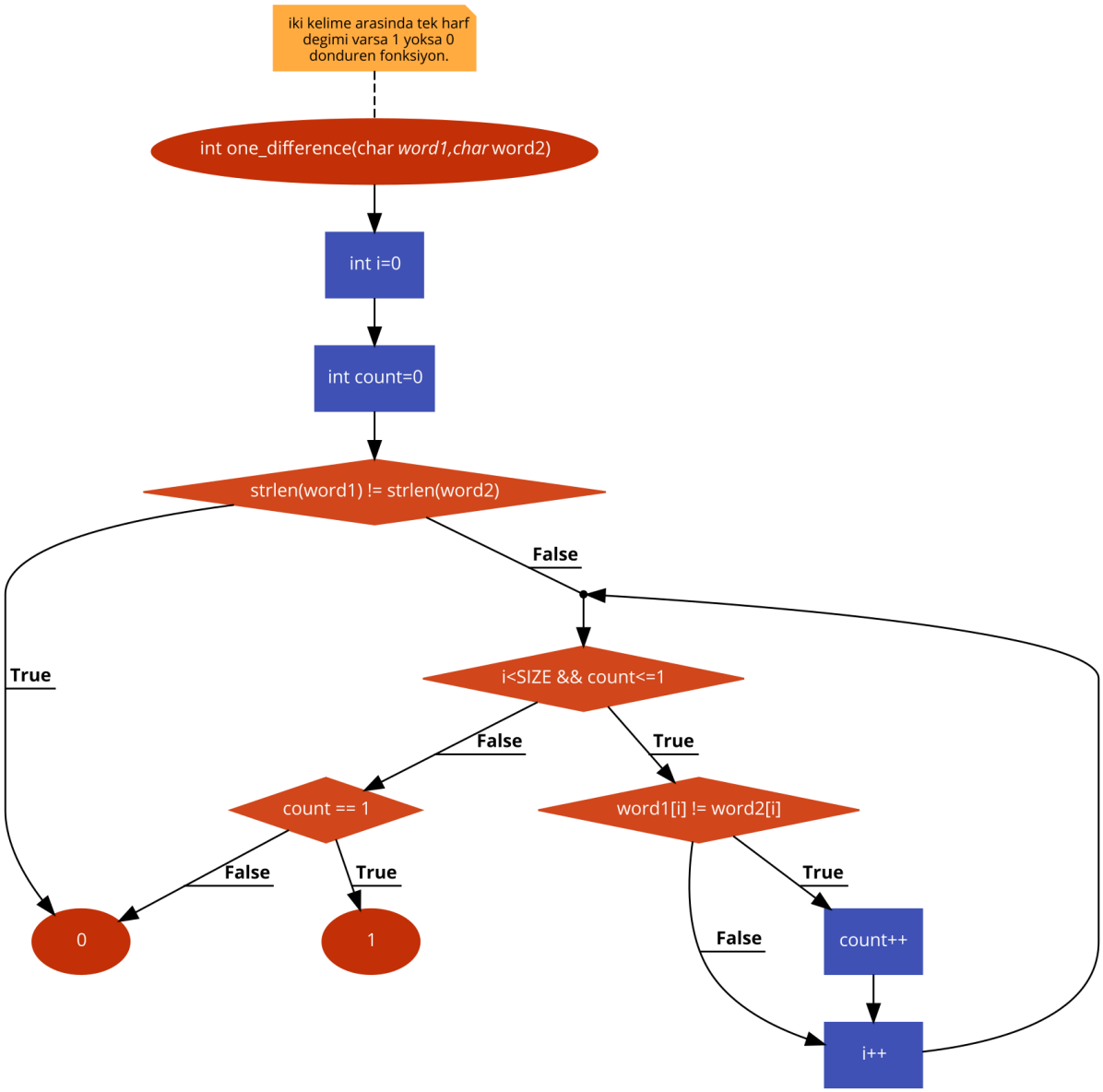

kuyruga eleman ekleyen fonksiyon
void enqueue(int data)
rear == NULL
True
False
rear = (struct node)malloc(1sizeof(struct node))
temp=(struct node)malloc(1sizeof(struct node))
rear->ptr = NULL
rear->ptr = temp
rear->info = data
temp->info = data
temp->ptr = NULL
front = rear
rear = temp
count++


kuyruk bos mu degil mi kontrol eden fonksiyon
int isEmpty()
(front == NULL) && (rear == NULL)
True
False
1
0


komsuluk matrisinin ilk deger atamasini yapan fonksiyon
void initialize_adjacency(ADJACENCY words,int size)
int i
i=0
i<size
True
(words)[i].neighbours=Allocator(1)
i++


int icin allocation yapan fonksiyon
int Allocator(int size)
int array
array=(int)malloc(sizeof(int)size)
array
False
True
printf("Array Not Allocated !!! Quitting...")
array
0


int diziler icin realloc yapan fonksiyon
void re_allocator(int arr,int size)
arr=(int)realloc(arr,(sizeof(int)(size)))
(*arr
False
True
)
printf("Not re-allocated... Quitting...")
exit(0)


iki kelime arasinda tek harf degimi varsa 1 yoksa 0 donduren fonksiyon.
int one_difference(char word1,char word2)
int i=0
int count=0
strlen(word1) != strlen(word2)
True
False
i<SIZE && count<=1
False
True
count == 1
word1[i] != word2[i]
False
True
True
False
0
1
count++
i++

2.UYGULAMA

ANALİZ

1) Örnek 1 -DÖNÜŞÜM VAR

word1=withy(2371)	word2=witty(2372)

ITER	STEP	DEQ INDEX	DEQ WORD	ENQ INDEX	ENQ WORDS	TMP_ARRAY	found	isEmpty
1	1	2731	withy	-	-	2371	0	0
1	2	-	-	1518	pithy	2371,1518	0	0
1	3	-	-	2367	wishy	2371,1518,2367	0	0
1	4	-	-	2370	withe	2371,1518,2367,2370	0	0
1	5	-	-	2372	witty	2371,1518,2367,2370,2372	0	0
2	6	1518	pithy	-	-	2371,1518,2367,2370,2372	0	0
3	7	2367	wishy	-	-	2371,1518,2367,2370,2372	0	0
3	8	-	-	799	fishy	2371,1518,2367,2370,2372,799	0	0
3	9	-	-	2326	washy	2371,1518,2367,2370,2372,799,2326	0	0
3	10	-	-	2368	wispy	2371,1518,2367,2370,2372,799,2326,2368	0	0
4	11	2370	withe	-	-	2371,1518,2367,2370,2372,799,2326,2368	0	0
4	12	-	-	1216	lithe	2371,1518,2367,2370,2372,799,2326,2368,1216	0	0
4	13			2177	tithe	2371,1518,2367,2370,2372,799,2326,2368,1216,2177	0	0
5	14	2372	witty	-	-	2371,1518,2367,2370,2372,799,2326,2368,1216,2177	1	0

withy→witty

2) Örnek 2 -DÖNÜŞÜM YOK

word1=villa(2298)	word2=whale(2340)

ITER	STEP	DEQ INDEX	DEQ WORD	ENQ INDEX	ENQ WORDS	TMP_ARRAY	found	isEmpty
1	1	2298	villa	-	-	2298	0	0
1	2	-	-	2300	viola	2298-2300	0	0
2	3	2300	viola	-	-	2298-2300	0	0
3	4	-	-			2298-2300	0	1

3) Örnek 3- DÖNÜŞÜM VAR

word1=abide(7)	word2=above(11)

									QUEUE
ITER	STEP	DEQ INDEX	DEQ WORD	ENQ INDEX	ENQ WORDS	TMP_ARRAY	found	isEmpty	7
1	1	7	abide	-	-	7	0	0	
1	2	-	-	8	abode	7,8	0	0	8
1	3	-	-	81	amide	7,8,81	0	0	8,81
1	4	-	-	127	aside	7,8,81,127	0	0	8,81,127
2	5	8	abode	-	-	7,8,81,127	0	0	81,127
2	6	-	-	11	above	7,8,81,127,11	0	0	81,127,11
2	7	-	-	105	anode	7,8,81,127,11,105	0	0	81,127,11,105
3	8	81	amide	-	-	7,8,81,127,11,105	0	0	127,11,105
4	9	127	aside	-	-	7,8,81,127,11,105	0	0	11,105
5	10	11	above	-	-	7,8,81,127,11,105	1	0	105

abide→abode→above

SOURCE CODE

```
/**
@file

Verilen dosyadaki kelimelerden secilen iki kelime arasindaki donusumu graph ve kuyruk yapisi kullanarak bulan program
Not : Program buyuk-kucuk harfe duyarlidir.

@author

Name             :      Muhammed Yasin SAGLAM
Student No       :      15011804
Date             :      05/05/2017
E-Mail           :      myasinsaglam1907@gmail.com
Compiler Used    :      GCC
IDE              :      DEV-C++(Version 5.11)
Operating System :      Windows 10 educational edition
*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define SIZE 5

//Optimizasyona uyumlu komsuluk matrisi tanimlaniyor
typedef struct {
    int *neighbours; //komsularin indislerinin tutuldugu dizi
    int neighbour_count; //komsu sayisi
}ADJACENCY;

//kuyruk yapisi tanimlaniyor
struct node{
    int info; //kuyruk node una  gonderilecek indis degerini tutacak olan degisken
    struct node *ptr; //bir sonraki kuyruk node unun adresini tutan degisken
}*front,*rear,*temp,*front1;

int count = 0; //kuyrukta bulunan eleman sayisini tutan degisken

//KULLANILAN FONKSIYONLARIN PROTOTIPLERI
int one_difference(char *word1,char *word2);
void control_for_lab();
int *Allocator(int size);
void re_allocator(int** arr,int size);
void initialize_adjacency(ADJACENCY **words,int size);
int front_element();
void create();
void dequeue();
void enqueue(int value);
int isEmpty();

int main(){

    //control_for_lab();  //labda istenecek olan kontrol icin yazilan fonksiyon

    int line=0; //satir sayisini tutan degisken, satir sayisi kadar kelime bulunuyor
    int i,j,k; //indis degiskenleri
    int queue_size=0; //kuyruk boyutunu tutan degisken
    int choice=1; //programi sonlandiran degisken
    int *tmp_arr = Allocator(1); //kuyruk islemleri icin yedek dizi olusturuluyor. Kuyruga giren dugumlerin indislerini sadece 1 kez tutacak ol
    char **list; //tum kelimeleri tutacak olan dizinin deklerasyonu
    ADJACENCY *words; //her bir kelimenin komsu sayilarini ve komsularinin indislerini tutan yapi deklerasyonu
    FILE *fp ; //file pointer

    fp=fopen("kelime.txt","r"); //dosya okuma modunda aciliyor
    if(!fp){ //kontrol
        printf("file error....");
        exit(0);
    }


    list=(char**)malloc(sizeof(char*));  //5 harflik kelimeleri tutan dinamik dizi olusturuluyor
    *list=(char*)malloc(sizeof(char)*SIZE);
        if(!list){
            printf("allocation error!!! Quitting...");
            exit(0);
        }

    while(!feof(fp)){
        fscanf(fp,"%s",list[line]);  //kelimeler dosyadan diziye okunuyor
        line++; //Satir sayisi dinamik kelime dizisini genisletebilmek icin her okumadan sonra arttiriliyor
        list=(char**)realloc(list,sizeof(char*)*(line+2)); //kelime dizisi genisletiliyor
        list[line]=(char*)calloc(sizeof(char),SIZE);
    }

    words = (ADJACENCY*)calloc(sizeof(ADJACENCY),line); //her bir kelimenin komsu sayilarini ve komsularinin indislerini tutan yapi icin kelime
        if(!words){
            printf("allocation error!!! Quitting...");
            exit(0);
        }

    //KOMSULUK YAPISI OLUSTURULUYOR
    initialize_adjacency(&words,line); //komsuluk yapisinin ilk deger atamalari yapiliyor
    k=0; //diziyi genisletmede kullanilan komsuluk indislerinin takibini saglayan indis degiskeni (anlasilabilirlik ve erisim suresini azaltmak
    for(i=0;i<line;i++){
    //  printf("\n%d. %s ==> ",i,list[i]);      //kontrol amacli yazdirma ilk kelimeyi yazdirir
        for(j=0;j<line;j++){
            if(one_difference(list[i],list[j])){ //iki kelime arasinda bir harf degismis ise
                k++;    //ilk kelimenin komsu sayisini arttir
                re_allocator(&words[i].neighbours,k); //komsuluk dizisini komsu sayisi kadar genislet
```

```
                words[i].neighbours[k-1]=j; //komsu kelimenin indisini ilk kelimenin komsuluk dizisine ekle
                words[i].neighbour_count++; //yapi icerisindeki komsu sayisini arttir
    //          printf(" %d- %s, ",words[i].neighbours[k-1],list[words[i].neighbours[k-1]]);    //kontrol amacli yazdirma ilk kelime icin bulun
            }
        }
        k=0; //yeni kelimeye gecildiginde global komsu indisini tutan degiskeni sifirla
    //  printf("\nneighbour_count %d \n",words[i].neighbour_count);     //kontrol amacli yazdirma komsu sayisini yazdirir
    }

    //kullanicidan alinacak kelimeler icin yer ayriliyor
    char *word1=(char*)malloc(sizeof(char)*SIZE);
    char *word2=(char*)malloc(sizeof(char)*SIZE);
        if(!word1 && !word2){
            printf("Allocation error!!! Quitting....");
            exit(0);
        }
while(choice!=0){
    int found=0;    //aranan kelimelerin verilen sette olup olmadigini kontrol eden degisken
    int index1=-1; //indisi aranan ilk kelimenin indisini tutan degisken
    int index2=-1; //indisi aranan ikinci kelimenin indisini tutan degisken
        i=0;
    while(!found){ //verilen set icinde girilen kelimeler araniyor
        printf("\nEnter first word(source) : ");
        scanf("%s",word1);
        printf("\nEnter second word(destination) : ");
        scanf("%s",word2);
        while(i<line){ //verilen kelimelerin kelime.txt dosyasininda bulunup bulunmadiklarina bakiliyor... Varsa indisleri bulunacaktir.
            if(strcmp(word1,list[i])==0)
                index1=i; //bulunan ilk kelimenin indisi ataniyor
            if(strcmp(word2,list[i])==0)
                index2=i; //bulunan ilk kelimenin indisi ataniyor
            i++;
            if(index1!=-1 && index2!=-1){ //eger iki kelimede sette bulunmussa cikiliyor
                found=1;
            }
        }
        i=0;
        if(!found){ //kelimeler bulunamazsa uyari yazdirilir ve tekrar okunur
            system("CLS"); //ekran temizlenir
            printf("\n*******PLEASE ENTER WORDS IN GIVEN TEXT FILE***********\n");
        }
    }
    //okunan kelimelerin indisi bulunduktan sonra kelimeler icin ayrilan alan serbest birakilir.

    printf("\nindexes of word1 is %d and word2 is %d \n",index1,index2); //kontrol amacli yazdirma //bulunan kelimelerin indislerini yazdirir
    create();

    //indis degiskenleri sifirlaniyor.
    i=0; //komsuluk dizilerinin cevrim degiskeni
    j=0; //yedek dizinin cevrim degiskeni
    int step_size=0; //kuyrugun kacinci elemanina gelindigini tutar
    int tmp_size=1; //yedek dizinin boyutunu tutacak olan degisken
//  k=index1;       //cevrim indisi olan k ya ilk kelimenin indisi baslangic olarak atanir
    found=0;        //donusumun tamamlanma durumunu kontrol eder
    enqueue(index1);    //ilk kelimenin indisi kuyruga atilir
    tmp_arr[0]=index1;  //ilk elemanin indisi yedek diziye de atilir
    int control_add;    //daha once kuyruga elemanin eklenip eklenmeme durumunu kontrol eden degisken
    while(!isEmpty() && found==0){ //kuyruk doluysa ve 2. kelimeye donusum olmamissa
        k=front_element();  //cevrim indisi olan k ya ilk kelimenin indisi baslangic olarak atanir
        dequeue();  //ilk node kuyruktan cekilir
        step_size++; //adim sayisi artirilir
    //  printf("\nstep size : %d\n",step_size); //kontrol amacli yazdirma
    //  printf("%d-cekilen -->%s\n",k,list[k]); //kontrol amacli yazdirma
        if(index2 == k){ //eger k degeri donusturulecek kelimenin indisine esitse
            found=1; //donusum tamamlanmis demektir
        }
        else{ //eger esit degilse
            for(i=0;i<words[k].neighbour_count;i++){ //tum komsu nodelar taraniyor
                j=0; //yedek dizi indisi sifirlanir
                control_add=1; //ekleme gerekiyor
                while(j<tmp_size && control_add){ //kuyruga daha once eklenen ayni node var mi yok mu indisler uzerinden kontrol ediliyor
                    if(words[k].neighbours[i] == tmp_arr[j]){  //eger daha once eklenmisse
                        control_add=0; //ekleme durumunu sifirlar
                    }
                    j++; //cevrime devam edilir
                }
                if(control_add){ //ekleme gerekiyorsa
                    //  printf("--------->%d eklenen -->%s\n",words[k].neighbours[i],list[words[k].neighbours[i]]); //kontrol amacli yazdirma
                        enqueue(words[k].neighbours[i]); //kuyruga ekleniyor
                        re_allocator(&tmp_arr,tmp_size+1);  //yedek dizinin yeni boyutu icin memory de yer ayriliyor
                        tmp_arr[tmp_size]=words[k].neighbours[i]; //yedek dizinin son elemanina kuyruga eklenen nodelarin indisleri ataniyor
                        tmp_size++; //tutulan yedek dizinin boyutu artiriliyor
                }
            /* //kontrol amacli yazdirma
            int z;
            printf("\n");
            for(z=0;z<tmp_size;z++){
                printf(" %d ",tmp_arr[z]);
            }
            system("PAUSE");    */ //PRINT TO CONTROL
            }
            //system("PAUSE");
        }
    }

    //Sonuc ekrana yazdiriliyor
```

```
    if(found==1){
        printf("\n\n---------------TRANSFORMATION FOUND------------------\n\n");
            //printf("\nstep size : %d\n",step_size);  //adim sayisi ekrana yazdiriliyor

    i=0; //indis degiskeni sifirlaniyor
    j=tmp_size; //ust degeri tutacak olan indis degiskeni
    int *path_arr=Allocator(1);
    int path_size=1;
    path_arr[0]=index2;
    printf("PATH IS : \n");
//  printf(" %s ",list[index2]);
    char* temp = list[index2];
    while(i<j){
        if(one_difference(temp,list[tmp_arr[i]])){
            //printf(" %s ",list[tmp_arr[i]]);
            temp=list[tmp_arr[i]];
            re_allocator(&path_arr,path_size+1);
            path_arr[path_size]=tmp_arr[i];
            path_size++;
            j=i;
            i=0;
        }
        else{
            i++;
        }
    }

    for (i=path_size-1; i>0; i--){
        printf(" %s -->", list[path_arr[i]]);
    }
    printf(" %s...",list[path_arr[i]]);
    printf("\n\nTotal Transform Number is : %d \n",path_size-1);
    }
    else{
        printf("\n\n-----------------NO TRANSFORMATION--------------\n\n");
    }
    printf("\nEnter 0 to exit, 1 to continue \n");
    scanf("%d",&choice);
    system("CLS");
}
    printf("GOOD BYE!!!");
    fclose(fp);
    free(list);
    free(words);
    free(word1);
    free(word2);
    free(tmp_arr);
    return 0;
}

//komsuluk matrisinin ilk deger atamasini yapan fonksiyon
void initialize_adjacency(ADJACENCY **words,int size){
    int i;
    for(i=0;i<size;i++){
        (*words)[i].neighbours=Allocator(1);
    }
}

//iki kelime arasinda tek harf degimi varsa 1 yoksa 0 donduren fonksiyon.
int one_difference(char *word1,char *word2){
    int i=0;
    int count=0;
    if(strlen(word1) != strlen(word2))
        return 0;

    while(i<SIZE && count<=1){
        if(word1[i] != word2[i])
            count++;
            i++;
    }
    if(count == 1 )
        return 1;
    else
        return 0;
}

//labda istenecek olan ve komsuluk matrisinin dogrulugunu ispatlayan fonksiyon
void control_for_lab(){
    char *word1=(char*)malloc(sizeof(char)*SIZE);
    char *word2=(char*)malloc(sizeof(char)*SIZE);
        if(!word1 && !word2){
            printf("\nAllocation error!!! Quitting...\n");
            exit(0);
        }
    printf("\nEnter first word(source) : ");
    scanf("%s",word1);
    printf("\nEnter second word(destination) : ");
    scanf("%s",word2);

    if(one_difference(word1,word2)==1)
        printf("\nYES\n");
    else
        printf("\nNO\n");
    free(word1);
    free(word2);
    system("PAUSE");
```

```
    exit(0);
}

//int diziler icin realloc yapan fonksiyon
void re_allocator(int** arr,int size){
    *arr=(int*)realloc(*arr,(sizeof(int)*(size)));
        if(!(*arr)){
            printf("Not re-allocated... Quitting...");
            exit(0);
        }
}

//int icin allocation yapan fonksiyon
int *Allocator(int size){
    int *array;
    array=(int*)malloc(sizeof(int)*size);
        if(!array){
            printf("Array Not Allocated !!! Quitting...");
            return 0;
        }
    return array;
}

//kuyruk bos mu degil mi kontrol eden fonksiyon
int isEmpty(){
    if((front == NULL) && (rear == NULL))
        return 1;
    else
        return 0;
}

//kuyruga eleman ekleyen fonksiyon
void enqueue(int data){

    if (rear == NULL)
    {
        rear = (struct node *)malloc(1*sizeof(struct node));
        rear->ptr = NULL;
        rear->info = data;
        front = rear;
    }
    else
    {
        temp=(struct node *)malloc(1*sizeof(struct node));
        rear->ptr = temp;
        temp->info = data;
        temp->ptr = NULL;

        rear = temp;
    }
    count++;
}

//kuyruktan eleman silen fonksiyon
void dequeue(){
front1 = front;

    if (front1 == NULL)
    {
        printf("\n Error: Trying to display elements from empty queue");
        return;
    }
    else
        if (front1->ptr != NULL)
        {
            front1 = front1->ptr;
            //printf("\n Dequed value : %d", front->info);
            free(front);
            front = front1;
        }
        else
        {
         //   printf("\n Dequed value : %d", front->info);
            free(front);
            front = NULL;
            rear = NULL;
        }
        count--;
}

//kuyruk olusturan fonksiyon
void create(){
    front = rear = NULL;
}

//kuyrugun ilk elemanini disari donduren fonksiyon
int front_element(){
    if ((front != NULL) && (rear != NULL))
        return(front->info);
    else
        return 0;
}
```