VERİ YAPILARI VE ALGORİTMALAR

- ÖDEV 1 -

GRUP 1

İrem ATILGAN

17061036

04.03.2020

Konu : Cache Buffer (Önbellek Tamponu) Tasarımı

Problem: Bu ödevde, double linked list (ikili linkli liste) veri yapısını kullanarak bir cache buffer sistemi tasarlamanız ve gerçeklemeniz istenmektedir. Gerçekleştireceğiniz cache buffer, bir network ortamında yapılan web sayfası isteklerini, hızlı ve kaliteli hizmet verebilmek için depolamak ve servis için önceliklendirmek amacı ile kullanılacaktır.

Ödev 4 ana bölümden oluşmaktadır:

1. Double linked list düğümlerinin ve veri yapısının gerçeklenmesi: Her bir düğümde; istenen sayfanın adresi, toplam istek sayısı (sayacı), önceki ve sonraki düğümlerin işaretçileri tutulmalıdır. Düğümler üzerinde gezinti, sayaç sorgulama, düğüm ekleme ve silme gibi işlemler gerçekleştirilmelidir.

2. Verilen işlemin double linked list kullanılarak çözümü:

a. Giriş bilgisinin okunması:

i. İstek yapılan sayfaları string olarak okuyunuz (Ör: ABC, ABE, BCD, …). Sayfalar bir metin belgesinden satır satır da okunabilmelidir.

b. Girdinin işlenmesi:

i. Girilen sayfa (veya dosyadan okunan 1 satır) önceden cache buffer'da yoksa yeni bir düğüm olarak yaratılır, sayacı 1 yapılır ve başlangıç düğümü (head node) olarak cache buffer'a eklenir.

ii. Girilen sayfa cache buffer'da mevcut ise, ilgili düğümün sayacı 1 arttırılır. Sayaç belirlenen bir eşik değerinin (Ör: T = 10) üzerine geçerse, bu düğüm cache buffer'da head node olarak atanır. (Önceki ve sonraki düğümlerin işaretçileri de düzenlenir.)

c. Cache buffer'ın bakımı:

i. Cache buffer'ın belirli bir kapasitesi vardır (Ör: L=20). Bu kapasite aşılırsa, cache buffer'ın son elemanı (düğümü) silinir (2.b maddesinde kontrol edilmelidir).

3. Cache buffer'ın hizmet vermesi: Her yeni istek geldiğinde, 2.b maddesi işletilir ve cache buffer'ın mevcut hali linkli liste şeklinde, düğümlerin içerikleri ile (adres ve sayacı) ekranda gösterilir. 4. Cache Buffer'ın Temizlenmesi: Kullanıcıya listeleme işleminden sonra cache'i temizlemek isteyip istemediği sorulmalıdır. Temizlemeye onay verilirse listedeki tüm elemanlar silinmelidir.

4. Cache Buffer'ın Temizlenmesi: Kullanıcıya listeleme işleminden sonra cache'i temizlemek isteyip istemediği sorulmalıdır. Temizlemeye onay verilirse listedeki tüm elemanlar silinmelidir.

GİRİŞ

Problemin çözülmesinde 11 fonksiyon tasarlanmış ve bir main fonksiyonu kullanılmıştır. Program genel olarak şu şekilde işlemektedir (Gelecek bölümde daha ayrıntılı açıklanacaktır) :

1. Main fonksiyonunda kullanıcıdan eşik ve kapasite değerleri alınır.
2. Giriş yapılacak node değerlerinin dosyadan mı yoksa konsoldan mı alınacağı kullanıcıya sorulur.
3. Cevaba bağlı olarak (**scanFile, openFile**) veya **scanConsole** fonksiyonları çalıştırılır
4. **scanFile** Fonksiyonu :
    a. Head node **initHead()** fonksiyonu ile oluşturulur
    b. Kapasitenin dolu olup olmadığı **checkCapacity()** fonksiyonuyla öğrenilerek mevcut programdan çıkılır ya da head node'un gösterdiği adrese değer atanarak çalışmaya devam edilir.
    c. Dosyadaki tüm değerler okunana dek,
        i. Her değer teker teker alınır ve mevcut double linked list'te olup olmadıkları kontrol edilir.
        ii. Eğer mevcut listede değillerse önce **checkCapacity()** ile mevcut listenin yeni gelen node ile kapasitesinin aşılıp aşılmadığı kontrol edilir. Aşılmışsa son node silmek üzere **deleteNode()** fonksiyonu çağrılır. Aşılmamışsa bir işlem uygulanmaz.
        iii. Son olarak yeni gelen değer için **pushNode()** fonksiyonu ile listenin başına yeni bir node getirilir. Bu node'a değer olarak daha önce okumuş olduğumuz string değeri aktarılır.
        iv. Son durum kullanıcıya gösterilmek üzere **listNodes()** fonksiyonu ile yazdırılır.
    d. Head node'un adresi main fonksiyonuna return edilir.
5. **scanConsole** Fonksiyonu:

    scanFile() fonksiyonundan çok farklı olmasa da bazı konularda değişiklik göstermektedir.

    a. Kullanıcıya kaç tane string gireceği sorulur.
    b. Cevaba bağlı olarak char dizisi yaratılır.
    c. **scanFile()** fonksiyonundaki gibi listenin kapasitesi kontrol edilir.
    d. Girilecek stringler bitene dek **scanFile()** fonksiyonundaki i,ii,iii,iv ve d maddeleri uygulanır.
6. **checkOF** Fonksiyonu :
    a. Kendisine parametre olarak gönderilen node'un NULL'a eşit olup olmadığını kontrol eder. Eşitse programdan çıkar, değilse normal akışına devam eder.
7. **scanFile/scanConsole** fonksiyonu main fonksiyonuna döndükten sonra kullanıcıya cache buffer'ı temizlemek isteyip istemediği sorulur. Kullanıcının isteğine bağlı olarak **freeNodes()** fonksiyonu çağrılır.
8. **freeNodes** fonksiyonu :
    a. Double Linked List'teki tüm node'ları baştan sonra gezerek bellekten siler.

PROGRAM

0.struct

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

typedef struct NODE{
    int count;
    char value[50];

    struct NODE* next;
    struct NODE* prev;


}NODE;
```

1.main

```c
int main()
{
    char ans;
    int file_or_console;
    NODE* head;

    int T,L;
    printf("Esik Degeri Giriniz (T) = ");scanf("%d",&T);
    printf("Kapasite Giriniz (L) = ");scanf("%d",&L);
    printf("\n");

    printf("Dosya ile giris yapmak icin 1, Konsoldan giris yapmak icin 0 giriniz : ");scanf("%d",&file_or_console);

    if(file_or_console)             //Dosya ile giriş yapılacaksa
    {
        FILE* fp = openFile();
        head = scanFile(fp,T,L);
        fclose(fp);
    }
    else                            //Konsoldan giriş yapılacaksa
        head = scanConsole(T,L);


    printf("\nCache Buffer'i temizlemek istiyor musunuz ? [y\\n] ");
    scanf(" %c",&ans);
    if(ans == 'y')
        freeNodes(&head);           //Kullanıcı cache buffer'ı temizlemek isterse


    return 0;
}
```

Program ilk olarak main fonksiyonundan başlar.

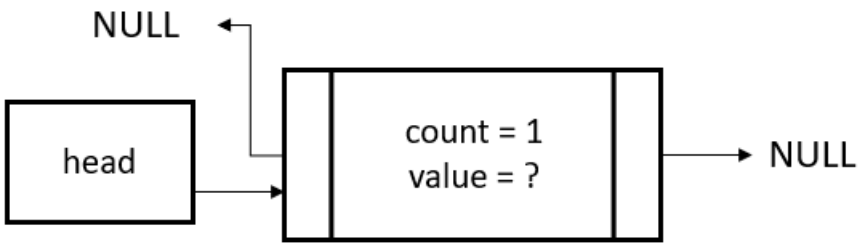
- Double Linked List'te kullanılacak Eşik (T) ve Kapasite (L) değeri kullanıcıdan alınır.
- Kullanıcının string değerlerini nasıl gireceğinin bilgisi alınır ve file_or_console değişkeninde tutulur.
- Eğer değerler dosyadan kullanılacaksa openFile() fonksiyonu ile FILE* oluşturulur ve fp değişkenine atanır.
- Daha sonra dosyadaki değerleri okuması için scanFile() fonksiyonu çağrılır.
- Fonksiyon, oluşturulan linkli listenin head node'unun adresini döner.
- Açılmış olan dosya fclose() fonksiyonu ile kapatılır.
- Eğer değerler konsoldan alınacaksa scanConsole fonksiyonu çağrılır ve oluşturulan head node'un adresi main fonksiyonuna döner.
- Son olarak kullanıcıya cache buffer'ı temizlemek isteyip istemediği sorulur. Cevaba bağlı olarak freeNodes() fonksiyonu çağrılır ve listedeki tüm node'lar bellekten silinir.

2.initHEAD

```
//Double linked list'in head'ini oluşturmak için kullanılan fonksiyon
void initHead(NODE** head)
{
    (*head) = (NODE*)malloc(sizeof(NODE));
    if(checkOF((*head))) //head node oluşturulup oluşturulamadığını kontrol eder
    {
        //ilk node yaratıldığı için kendinden öncesini ve sonrasını gösteren NODE pointerlar NULL gösterir
        (*head)->next = NULL;
        (*head)->prev = NULL;
        (*head)->count = 1;
    }


}
```

initHead fonksiyonu, oluşturulan head node'un initialize etmek (başlangıç durumuna getirmek) için kullandığımız fonksiyondur.

- İlk olarak head node için bellekte NODE boyutunda yer ayrılır.
- Yerin başarılı bir şekilde ayrılabildiğini öğrenmek için checkOF() fonksiyonu çağrılır.
- Yer ayırma işlemi başarılıysa, head'in next'i ve prev'i NULL gösterir. Çünkü listede henüz sadece kendisi oluşturulmuştur.
- Node'a yeni bir string değeri yerleşeceği için sayaç (count) 1 yapılır.



3.checkOF

```
//Overflow olup olmadığını anlamak için kullanılan fonksiyon
int checkOF(NODE* node)
{
    if(node) //Eğer node NULL değerine sahip değilse overflow olmamış demektir
        return 1;
    else
    {
        printf("Hata!..Node olusturulamadi!..");
        exit(1);
    }


}
```

4.pushNode

Yeni gelen node'u Double Linked List'e eklemek için kullanılır. Ödevde bizden her yeni gelen elemanın yeni head olarak listenin **önüne** eklenmesi istendiği için fonksiyon bu yöntemle tasarlandı.

```
//Dosyadan okunan her yeni değeri Double Linked List'e yerleştirmek için kullanılan fonksiyon
void pushNode(NODE** head)
{
    //Yeni node için bellekte yer ayrılır
    NODE* newNode = (NODE*)malloc(sizeof(NODE));

    //Eğer node başarıyla oluşturulduysa
    if(checkOF(newNode))
    {
        newNode->next = (*head); //newNode'un next'i, head'in daha önce işaretmiş olduğu node'u,
        newNode->prev = NULL;    //newNode'un prev'i, NULL'u ve
        newNode->next->prev = newNode;  //newNode'dan sonra gelecek node'un prev'i, newNode'u gösterecek şekilde ayarlanır
        (*head) = newNode;   //Son olarak oluşturulan yeni node head yapılarak Linked List'in başına tamamıyla geçmiş olur
        newNode->count = 1;  //Node ilk kez oluşturulduğu için (kendiyle aynı değere sahip başka bir node olmadığından) sayacı 1 yapılır
    }

}
```

- İlk olarak oluşturulacak yeni node için bellekte yer ayrılır.
- Daha sonra bellekte başarılı bir şekilde yer ayrılıp ayrılmadığı kontrol edilmek üzere **checkOF** fonksiyonu çağrılır.
- Eğer bir problem yoksa,
  1. Yeni node head olacağı ve eski head node da listedeki **ikinci node** olacağı için :

*newNode->next = (*head);* yapılır

2. Head'in işaret edeceği yeni node'un gerisinde, değer içeren başka bir node olmayacağı için :

   newNode->prev = NULL; yapılır

3. Yeni node'dan sonra gelen node'un (eski head'in) önceki node olarak yeni node'u göstermesi için :

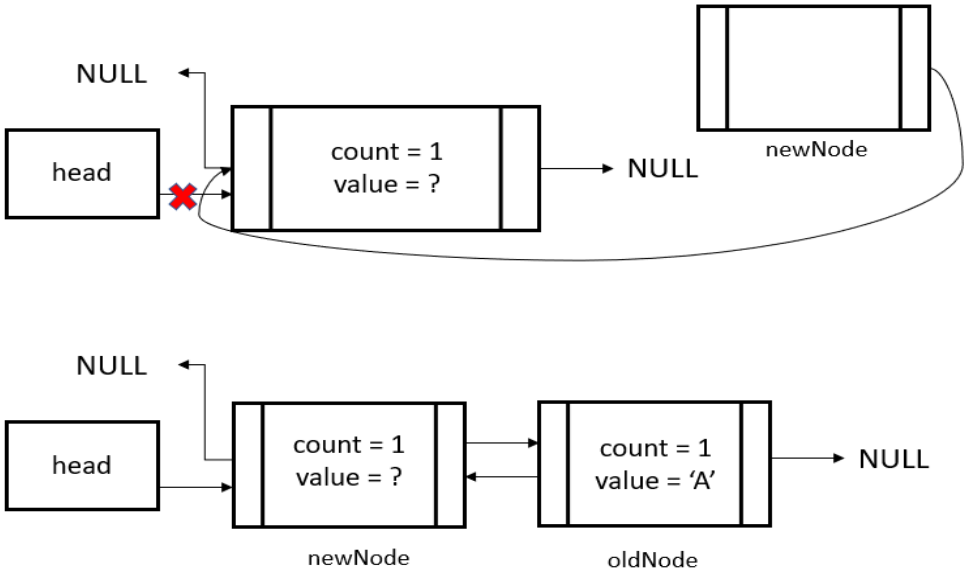
   *newNode->next->prev = (*head);* yapılır

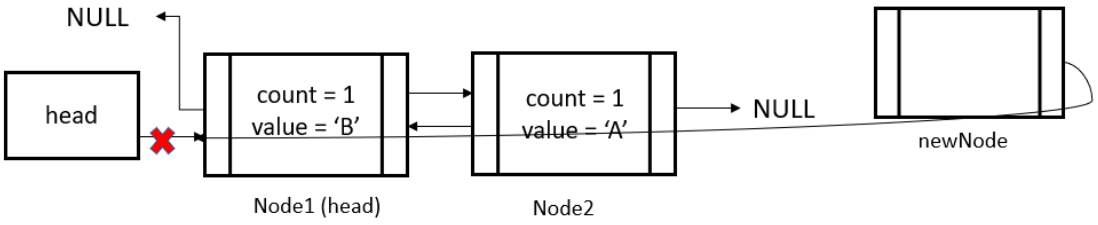
4. Artık head'in yeni node'u göstermesi gerektiğinden,

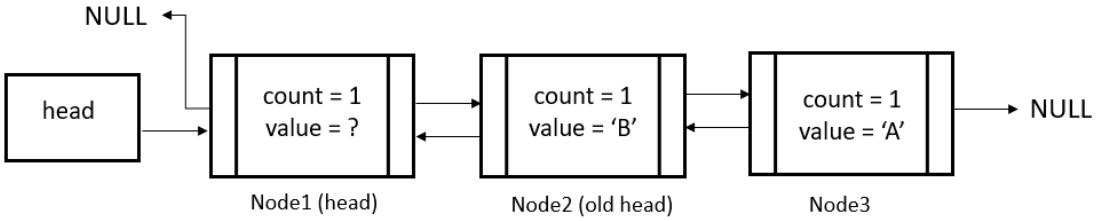
   *(*head) = newNode;* yapılır

5. Son olarak yeni node'a yeni bir değer ekleneceği için sayacı 1 yapılır.

Kısaca örnek bir süreç gösterilmek istenirse,







5.checkCapacity

```c
//Double Linked List'in kapasitesini aşıp aşmadığını kontrol eden fonksiyon
int checkCapacity(int L)
{
    static int capacity_counter = 0; //fonksiyonun her yeni elemanda artan kapasite değerini tutabilmesi için sayaç "static" tipinde yaratıldı
    capacity_counter++; //fonksiyon her çağrıldığında yeni bir node eklenmiş demektir
    if(L == 0) //Kullanıcının kapasiteyi sıfır girmesi durumunda program hata verir
    {
        printf("Linked list olusturulamaz!..");
        exit(1);
    }
    else if(capacity_counter > L) //Kapasite aşılmışsa başka bir fonksiyon yardımıyla listedeki son node silinecektir
    {
        capacity_counter--; //Bu yüzden dolan kapasite miktarı bir azaltılır
        return 1;
    }
    else
        return 0;  //Henüz kapasite aşılmamışsa

}
```

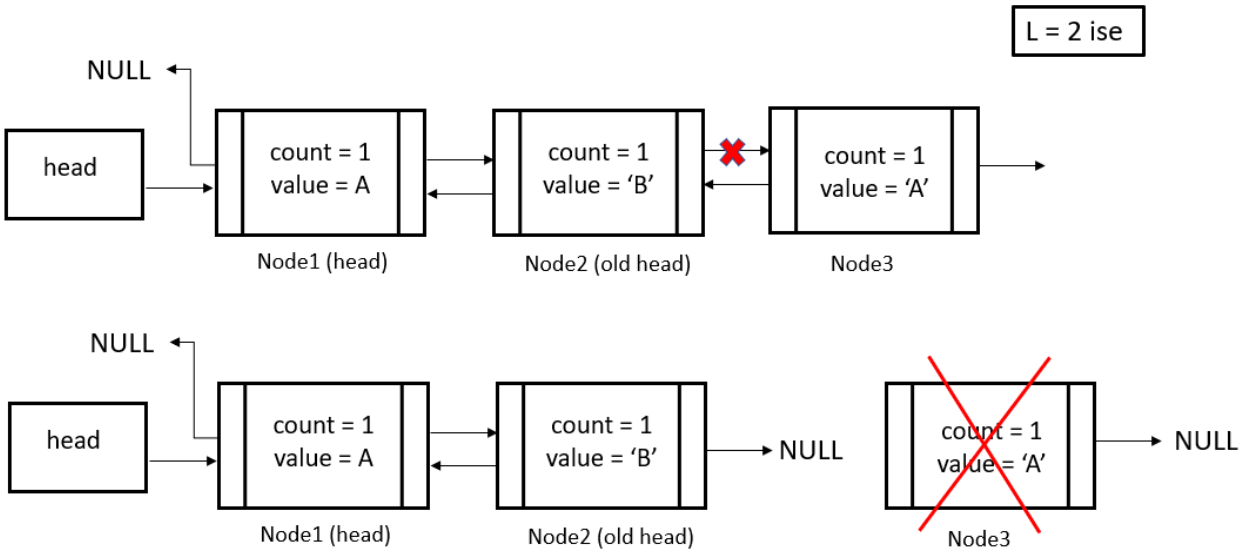
checkCapacity fonksiyonu kullanıcının bize şart koşmuş olduğu kapasitenin aşılıp aşılmadığını kontrol eder. Aşıldıysa 1, aşılmadıysa 0 döndürür. Burada dikkat edilmesi gereken nokta, kapasite sayacı capacity_counter'ın **static** olması. Bunun sebebi fonksiyonu her çağırdığımızda **eski doluluk oranını da elinde barındırabilmesi**. Çağırdığımız fonksiyonda değişkeni yaratıp adresini göndererek de değiştirebilirdik ancak böyle bir çözüm kodun **okunaklığı** ve fonksiyonun **esnekliği** açısından doğru bulundu.

6.deleteNode

```
//Kapasitenin aşılması durumunda çağırılan son node'u silme fonksiyonu
void deleteNode(NODE** head)
{
    NODE* current;
    for(current = *head;current->next != NULL;current = current->next); //Listenin sonuna gidilir

    current->prev->next = NULL; //Yeni son node, sondan bir önce gelen node olacağı için sondan bir önceki node'un next'i
                                //NULL gösterecek şekilde ayarlanır


    free(current); //Son node bellekten silinir


}
```

deleteNode fonksiyonu, linkli liste için verilen kapasitenin **aşılması** durumunda çağrılan fonksiyondur. Listedeki **son node'un** silinmesini sağlar.

- Son node'a gidilene dek listedeki tüm node'lar gezilir.
- Son node'a gelindiğinde, artık son node **kendinden bir önceki node** olacağı için, sondan ikinci node'un next'i **NULL** gösterilir.
- Son node **free** edilerek bellekten silinmiş olur.



7.searchNodes

```c
//Text dosyasından okunan her değerin listedeki elemanlardan biriyle aynı olup olmadığını öğrenmek için kullanılan fonksiyon
int searchNodes(NODE** head,char* string,int threshold)
{
    NODE* current = *head;
    NODE* tmp;

    //gezdiğimiz node'lardan biri, gelen stringle aynı olmadığı ve listeyi gezmeyi bitirmediğimiz sürece aramaya devam edilir
    while((current != NULL) && strcmp(current->value,string))
    {
        current = current->next;
    }

    if(current == NULL) //Eğer string Double Linked List'teki hiçbir elemanla aynı değere sahip değilse
    {
        return 1;
    }
    else
    {
        (current->count)++; //Listedeki elemanlardan biri ile aynı değere sahipse elemanın bulunduğu node'un sayacı 1 artar
        if(current->count > threshold) //Mevcut node'un sayacı, eşik değerini geçmişse node, head olmak üzere ayarlanır
        {
            tmp = *head;                //head saklanmak için tmp isimli NODE* da tutulur


            /*
            Eğer son node yeni head olacaksa
            Son node'dan önce gelen node artık sonuncu node olacağı için next'i NULL olur
            Ancak eşik değerini aşan node listenin son node'u değilse kendinden önce gelen node (prev)
            kendinden sonra gelen node'u (next) göstermelidir
            */

            current->prev->next = current->next;    //Bu kod satırı iki durum için de çalışmaktadır


            current->next = *head;      //head artık listedeki ikinci node olacağı için yeni head'in next'i eski head'i gösterir
            current->prev = NULL;       //mevcut node, head olacağı için prev'i artık NULL gösterir
            (*head)->prev = current;    //head'den önceki node, yeni head node'u gösterir
            *head = current;            //yeni head artık eşik değerini geçen node olmuştur

        }
        return 0;
    }


}
```

searchNodes fonksiyonunda dosyadan okunan/konsoldan okunan değerin ilgili linkli listede olup olmadığı kontrol edilir.

- Eğer listedeki hiçbir elemanla eşleşmiyorsa (**current == NULL**) yeni bir node oluşturmak üzere (pushNode) 1 döner.
- Eğer listedeki elemanlardan biri ile eşleşirse, eşleşilen node'un sayacı (current->count) bir arttırılır. Bu aşamadan sonra sayacı arttırılan node'un bize verilen eşik değerini (threshold) geçip geçmediğine bakılır.
- Eşik değeri geçilmişse mevcut node, head node yapılmak üzere,

1. Bulunduğumuz node'dan bir önceki node'un, bulunduğumuz node'dan bir sonraki node'u göstermesi sağlanarak current node aradan çıkarılır.

2. Mevcut head'in bulunduğumuz node'un sonraki elemanı olması için :
   current->next = *head; yapılır

3. Current node, head'in işaret edeceği node olacaktır. Bu yüzden kendinden bir önceki node'u gösteren *prev node pointer'ı NULL yapılır.

4. Head'in gösterdiği node artık ikinci node olacağı için *prev node pointer'ı birinci olacak olan current node'u gösterir.

5. Son olarak head'in, ilk node'u göstermesi gerektiğinden,
   (*head) = current; yazılır

8.listNodes

```c
//Double Linked List elemanlarını yazdırma fonksiyonu
void listNodes(NODE* head)
{
    NODE* current;
    current = head;

    printf("   %s , %d\t",current->value, current->count);
    for(current = head->next;current != NULL;current = current->next)
        printf("-> <-\t%s , %d\t\t",current->value, current->count);
    printf("\n");

}
```

9.openFile

```c
//Dosya açma fonksiyonu
FILE* openFile()
{
    FILE *fp;

    fp = fopen("input.txt","r");

    if(fp)
        return fp;
    else
    {
        perror("Dosya acilamadi!..");
        exit(1);
    }


}
```

10.scanFile

```
//Kullanıcı linked list elemanlarını txt dosyasından girmek isterse çalışacak fonksiyon
NODE* scanFile(FILE* fp,int threshold, int capacity)
{
    NODE* head;
    initHead(&head);    //head'e bellekte yer ayıran ve NODE pointer'larını düzenleyen fonksiyon
    char tmp[50];       //dosyadan okuyacağımız stringleri geçici olarak alan char dizisi

    //Head node oluşturulduğunda kapasitenin aşılıp aşılmadığı kontrol edilir
    if(checkCapacity(capacity))
    {
        deleteNode(&head); //Kapasite aşıldıysa node'u sil
    }


    fscanf(fp,"%s",head->value);  //Dosyadan okunan ilk değer head node'un işaret ettiği adresteki 'value' değişkenine yazılır
    listNodes(head);

    while(!feof(fp))
    {

        fscanf(fp,"%s",tmp);


        //Kullanıcıdan alınan veya dosyadan okunan string'in node'lardan birinde olup olmadığı kontrol edilir
        //Eğer hiçbir node ile aynı değere sahip değilse 1, aksi taktirde 0 döner
        if(searchNodes(&head,tmp,threshold))
        {
            //Listeye yeni bir node yerleşeceği için kapasitenin aşılıp aşılmadığı kontrol edilir
            if(checkCapacity(capacity))
                deleteNode(&head);  //Aşılmışsa son node'un silinmesi için deleteNode fonksiyonu çağrılır

            pushNode(&head);        //Yeni node'un listeye eklenmesi için pushNode fonksiyonu çağrılır

            strcat(head->value,tmp);    //Yeni head node'un gösterdiği adrese input olarak aldığımız string yerleştirilir

        }


        listNodes(head);


    }


    return head;    //Oluşan NODE pointer, main fonksiyonuna gönderilir

}
```

11.scanConsole

```
//Kullanıcı string değerlerini konsoldan girmek isterse
NODE* scanConsole(int threshold,int capacity)
{
    char inputs[50];
    short n;    //Giriş yapılacak string sayısı
    int i;

    printf("\nListeye kac eleman ekleyeceginizi giriniz : ");scanf("%d",&n);
    printf("\n");

    //Bu kısım scanFile fonksiyonundaki kodlarda yazılan mantıkla çalışmaktadır
    NODE* head;
    initHead(&head);
    if(checkCapacity(capacity)) //Kapasite kontrolü
    {
        deleteNode(&head);      //Kapasite aşılırsa node'u sil
    }

    scanf("%s",head->value);
    listNodes(head);

    for(i = 1; i < n; i++)
    {
        scanf("%s",inputs);

        if(searchNodes(&head,inputs,threshold)) //Girilen değere eşit node var mı diye kontrol et
        {
            if(checkCapacity(capacity))
                deleteNode(&head);

            pushNode(&head);                    //Eşit node bulunamamışsa listeye yeni node'u ekle

            strcat(head->value,inputs);
        }

       listNodes(head);
    }

    return head;
```

12.freeNodes

```
//Kullanıcının isterse cache buffer'ı temizleyebilmesi için kullanılan freeNodes fonksiyonu
void freeNodes(NODE** head)
{
    NODE* current = *head;
    for(current = (*head)->next; current->next != NULL; current = current->next)
        free(current->prev);


    free(current);

    printf("\nTum elemanlar temizlendi..");
}
```

Double Linked List yazdırıldıktan sonra kullanıcıdan **cache buffer'ı temizlemek isteyip istemediği bilgisi** alınır. Eğer kullanıcı temizlemek isterse freeNodes fonksiyonu çağrılır.

Bu fonksiyonda listedeki **2. Node'dan** başlanılır ve son node'a gelininceye kadar bulunduğumuz node'dan **bir önceki node** bellekten silinir.

Son olarak da listedeki **en son node** silinir.

EKRAN ÇIKTILARI

ÖRNEK 1 :

T = 2 L = 3

AB BA CY AB CY XYZ BA XYZ BA

```
Esik Degeri Giriniz (T) = 2
Kapasite Giriniz (L) = 3

Dosya ile giris yapmak icin 1, Konsoldan giris yapmak icin 0 giriniz : 1
   AB , 1
   BA , 1        -> <-   AB , 1
   CY , 1        -> <-   BA , 1          -> <-   AB , 1
   CY , 1        -> <-   BA , 1          -> <-   AB , 2
   CY , 2        -> <-   BA , 1          -> <-   AB , 2
   XYZ , 1       -> <-   CY , 2          -> <-   BA , 1
   XYZ , 1       -> <-   CY , 2          -> <-   BA , 2
   XYZ , 2       -> <-   CY , 2          -> <-   BA , 2
   BA , 3        -> <-   XYZ , 2         -> <-   CY , 2

Cache Buffer'i temizlemek istiyor musunuz ? [y\n] y

Tum elemanlar temizlendi..
--------------------------------
```

ÖRNEK 2 :

T = 3 L = 4

A B A AA BBB B A AB A B A BB

```
Esik Degeri Giriniz (T) = 3
Kapasite Giriniz (L) = 4

Dosya ile giris yapmak icin 1, Konsoldan giris yapmak icin 0 giriniz : 1
   A , 1
   B , 1        -> <-   A , 1
   B , 1        -> <-   A , 2
   AA , 1       -> <-   B , 1          -> <-   A , 2
   BBB , 1      -> <-   AA , 1         -> <-   B , 1          -> <-   A , 2
   BBB , 1      -> <-   AA , 1         -> <-   B , 2          -> <-   A , 2
   BBB , 1      -> <-   AA , 1         -> <-   B , 2          -> <-   A , 3
   AB , 1       -> <-   BBB , 1        -> <-   AA , 1         -> <-   B , 2
   A , 1        -> <-   AB , 1         -> <-   BBB , 1        -> <-   AA , 1
   B , 1        -> <-   A , 1          -> <-   AB , 1         -> <-   BBB , 1
   B , 1        -> <-   A , 2          -> <-   AB , 1         -> <-   BBB , 1
   BB , 1       -> <-   B , 1          -> <-   A , 2          -> <-   AB , 1

Cache Buffer'i temizlemek istiyor musunuz ? [y\n] y

Tum elemanlar temizlendi..
```