PROGRAMLAMA LABORATUVARI 2

1. PROJE

Mohamed Hosam Mohamed Gomaa Helwa, Issa Aljnadi
Kocaeli Üniversitesi, Bilgisayar Mühendisliği Bölümü
190201140@kocaeli.edu.tr, 200201120@kocaeli.edu.tr

***Özet* —Bu belge Programlama Laboratuvarı 1 der- sinin 2. Projesi için geliştirdiğimiz çözümü açıklama amacıyla hazırlanmıştır. Belgede farklı başlıklar altında projede belirtilen sorunun tanımı ile çözüm için kulla- nılan algoritmalar, kütüphaneler ve kodu yazmak ve derlemek için kullanılan araçlar açıklanmıştır. Kay- nakça belgenin en son başlığı olarak yeralmaktadır.**

***Anahtar Kavramlar* — Time Complexity, Space Complexity**

I. Giriş

Proje de ogrencilerden okunan kod Big O notasyonuna göre Zaman karmaşıklığının yer (Hafıza) karmaşıklığının ve kod çalıştırıldığında geçen süresin hesaplaması istenmiştir . Bu kod C dilinin sağladığı metotları kullanarak yazılmalıdır. Gelistirilecek olan kodun girdileri bir yazı dosyasından (input.txt) alınacaktır. Yada bu dosyanın adı ise program çalışırken kullanıcı tarafından verilmelidir . Algoritmanın Zaman ve Hafıza Karmaşıklığı Bu durumlar altında kullandığımız algoritmaların bize olan zaman ve hafıza maliyetlerini hesaplamak, bunlar hakkında bilgi sahibi olmak çok önemlidir.

Bu iki terim aslında beraber algoritmanın verimliliğini belirtiyor. İyi bir algoritmadan az yer kaplaması ve az zaman harcaması beklenir. yer (Hafıza) karmaşıklığı (Space Complexity) algoritmanın işlevini yerine getirmesi için kullandığı bellek miktarı olarak söylenebili.

Ornek N elemanlı bir dizi için kullanacağımız hafıza O(N) olacaktır. Algoritmanın işlevini yerine getirmesi için gereken bellek miktarına veya o bellek alanını veren bağıntıya alan maliyeti denir. Alan maliyeti program kodu, yığın ve veri için gerekli tüm alanları kapsar. S(n) şeklinde gösterilir.

Ek olarak, 4 bytelık tamsayı olan n elemanlı bir küme için alan maliyeti bağıntısı S(n)=4n şeklinde ifade edilmektedir (Space Complexity) Alan karmaşıklığı, bir algoritma veya işlemin girdi boyutuna göre çalıştırması gereken toplam bellek miktarını ölçer. Yani alan maliyetini ölçmek demektir diyebiliriz.

.

II. YÖNTEM

Projeye daha başlamadan önce bir kodu yazarken bir iş bölümü oluşturma kararı aldık. Bunun temel sebebi zaman kısıtlama- sıydı. Bu iş bölümüne uygun olarak birimiz ağaç veri yapısını oluşturma kısmıyla, birimiz de ağacın grafiksel olarak gösteril- mesi kısmıyla ilgilendi. Birbirinden iki ayrı süreç yerine sık sık birbiri ile kesişen ve biri öbürünü tamamlayan bir süreç yürüt- meye çalıştık.

III.A. *Araştırma*

Problemleri çözmek için farklı veri yapılarına ihtiyaç duyarız. Örneğin (Array, Stack, Matricies, C built in String Metodlar). Veya aynı veri yapısı üzerinde Array üzerinde sıralama için farklı farklı algoritmalar çalıştırabilirsiniz Örneğin (search string methods, Read files line by line methods) . tüm bu seçimlerimizin kodumuzun kalitesine, hızına, kullanışlılığına ve tükettiği kaynaklar yönünden sonuçları olur. Bundan dolayı yazılım geliştiricilerin bu konuyu iyi bir şekilde idrak etmesi önemlidir.

Biraz araştırmadan sonra, bu projeyi tamamlamak için, kodun Zaman karmaşıklığını elde etmek için metin dosyasını satır satır aramamız gerektiği sonucuna vardık. Bu bizim için zor oldu çünkü her türlü döngü ve özyinelemeli işlevi aramak çok fazla vakanız olduğundan ve tüm bu vakalara ulaşamayacağınız için çok zor. Bu yüzden çoğu durumda çalışacak bir algoritma elde etmek için elimizden gelenin en iyisini yapmaya çalıştık.

IV.B. *Uygulama*

1) ***Dosya Okuma***: Programın işleyiş sırasına göre hareket ettiğimiz için ilk uygulamaya koyulduğumuz kısım verillerin **input.txt** dosyasından okunması oldu.

Öncelikle kodda fonksiyonlarımız olup olmadığını anlamak için tüm satırları okudum ve ardından döngüleri ve fonksiyon çağrılarını aramak için ana fonksiyona ulaştım. Aşağıdaki sözde kodu takip ederek dosyaları satır satır sonuna kadar okumayı başarıyoruz.

While (dosyanın sonuna ulaşımına kadar)

İf (bir Foncsiyon adı bulundu zaman)

Onun karmaşıklığını hesapla ve bir yığına kaydet.

Else if (Bir for döngüsü varsa)

Onun karmaşıklığını hesapla ve yığına kaydet.

Else if (Bir while veya do while döngüsü varsa)

Onun karmaşıklığını hesapla ve yığına kaydet.

Else if (Bir int, double, char yada float varsa)

onun dizi olduğu olmadığını tanımlama.

Else

Kodun karmaşıklığını O(1)

End

2)*Döngünün karmaşıklığını hesaplama*:

Bir döngünün karmaşıklığını hesaplamak için, döngünün (n)'ye bağlı olduğu anlamına gelen bir değişkense döngünün bitiş değerini saptaması gerekiyordu ve sonra ++ veya -- operatörlerini saptadım.

If (for döngüsü bulunursa)

If(Döngü değişkene bağlı)

If ('++' yada '--' operatörler varsa)

Karmaşık O(n)

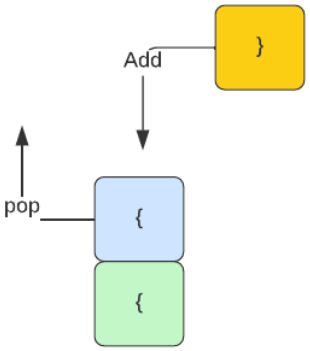
Else

Karmaşık O(1)

End If

3) Yığını Kullanma:

Yığını Stack, her while veya do while döngüsünün sonunu izlemek ve ardından "++" veya "--" operatörünün olup olmadığını kontrol etmek için içeri girmek için kullanılmıştır. kapalı bir küme parantez açık olana geldiğinde, açık olanın sınıftan çıkmasını sağlar.

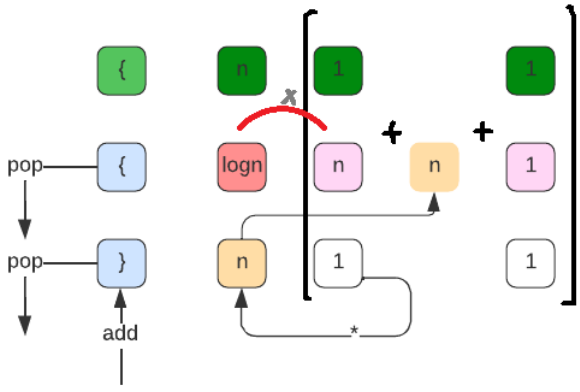


4) özyinelemeli fonksiyonlar:

Öncelikle text dosyasında main dışında fonksiyonlarımız olup olmadığını araştırıyoruz. Fonksiyonu bulduktan sonra, adının içinde çağrıldığını bulana kadar içini ararım. Fonksiyonu bulduktan sonra bir değişkene bağlı olarak çağırıp çağırmadığını kontrol ediyoruz.

5) iç içe döngüler:

Bir matris yardımıyla iç içe döngülerdeki toplam zaman karmaşıklığını hesaplamak için her döngünün başlangıcını ve sonunu bilmek ve yol boyunca her blok için toplam karmaşıklığı bulmak için kullanıldı.



Bir önceki resimde olduğu gibi, önce döngüleri kontrol ediyorum ve sonra diğer fonksiyonlarla karmaşıklığını alıyorum, ardından onları kendi küme parantezleri ile yığına ekliyorum. Bu parantez kapatıldığında, eleman sınıftan çıkıp matristeki konumunda önceki elemana çıkar. Matristeki öğeler, onlardan en büyük etkili öğeyi alarak karmaşıklık olarak bir araya getirilir.

6) Notlar :

Sayılar, hesaplamaları kolaylaştırmak için kodun karmaşıklığını temsil etmek için kullanılmıştır ve bu sayılar aşağıda listelenmiştir.

O(1) ⟶ 1
O(n) ⟶ 10
O(logn) ⟶ 3
O(2^n) ⟶ 313
O(n^2) ⟶ 100

V. Deneysel sonuçlar

Sonraki örnekler, programın farklı durumlar için test edilmesinin gerçek sonuçlarını temsil etmektedir.

1) *Logaritmik Karmaşıklık:*

index.txt - Notepad

File Edit Format View Help

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int count=10,i,n=10;
    int arr[n];
    i=1;
    do{
        arr[i]=i*count;
        printf("%d * %d = %d\n",i,count,i*count);
        i *= 2;
    }while(i<=n);
    return 0;
}
```

Ln 1, Col 1 130% Windows (CRLF) UTF-8

"F:\Computer-Engineering\Second Year\Second Semister\ProLab\Project1\pr...

```
space Complexity: n

Time Complexity: log(n)


Process returned 0 (0x0)   execution time : 0.035 s
Press any key to continue.
```

2) *ikinci dereceden karmaşıklık:*

index.txt - Notepad

File Edit Format View Help

```
#include <stdio.h>
int main(){
        int i,j;
        int sum = 0;
        int n=10;
        int arr[n][n];
        for (i=0;i<n;i++){
                for (j=0;j<n;j++){
                    arr[i][j]=i*j;
                        sum = sum + arr[i][j];
                }
        }
        printf("%d", sum);
        return 0;

}
```

Ln 1, Col 1 100% Windows (CRLF) UTF-8

"F:\Computer-Engineering\Second Year\Second Semister...

```
space Complexity: n^2

Time Complexity: n^2


Process returned 0 (0x0)   execution time : 0.191 s
Press any key to continue.
```

index.txt - Notepad

File Edit Format View Help

```
#include <stdio.h>
int main(){
        int i,j,k;
        int sum = 0;
        int n=10;
        int arr[n][n];
        for (i=0;i<n;i++){
            for (k=0;k<n;k++){
                for (j=0;j<n;j *= 2){
                    arr[i][j]=i*j;
                        sum = sum + arr[i][j];
                }
            }
        }
        printf("%d", sum);
        return 0;

}
```

Ln 1, Col 1 100% Windows (CRLF) UTF-8

"F:\Computer-Engineering\Second Year\Second Se...

```
space Complexity: n^2

Time Complexity: n^2*log(n)


Process returned 0 (0x0)   execution time : 0.037 s
Press any key to continue.
```

*index.txt - Notepad

File Edit Format View Help

```
#include <stdio.h>
int factorial (int n) {
    if (n <= 1)
        return 1 ;
    else
        return n * factorial(n-1);
}
int main(){
    int sonuc,n=6;
    sonuc=factorial(n);
    printf("%d", sonuc);
    return 0;

}
```

Ln 14, Col 2 100% Windows (CRLF) UTF-8

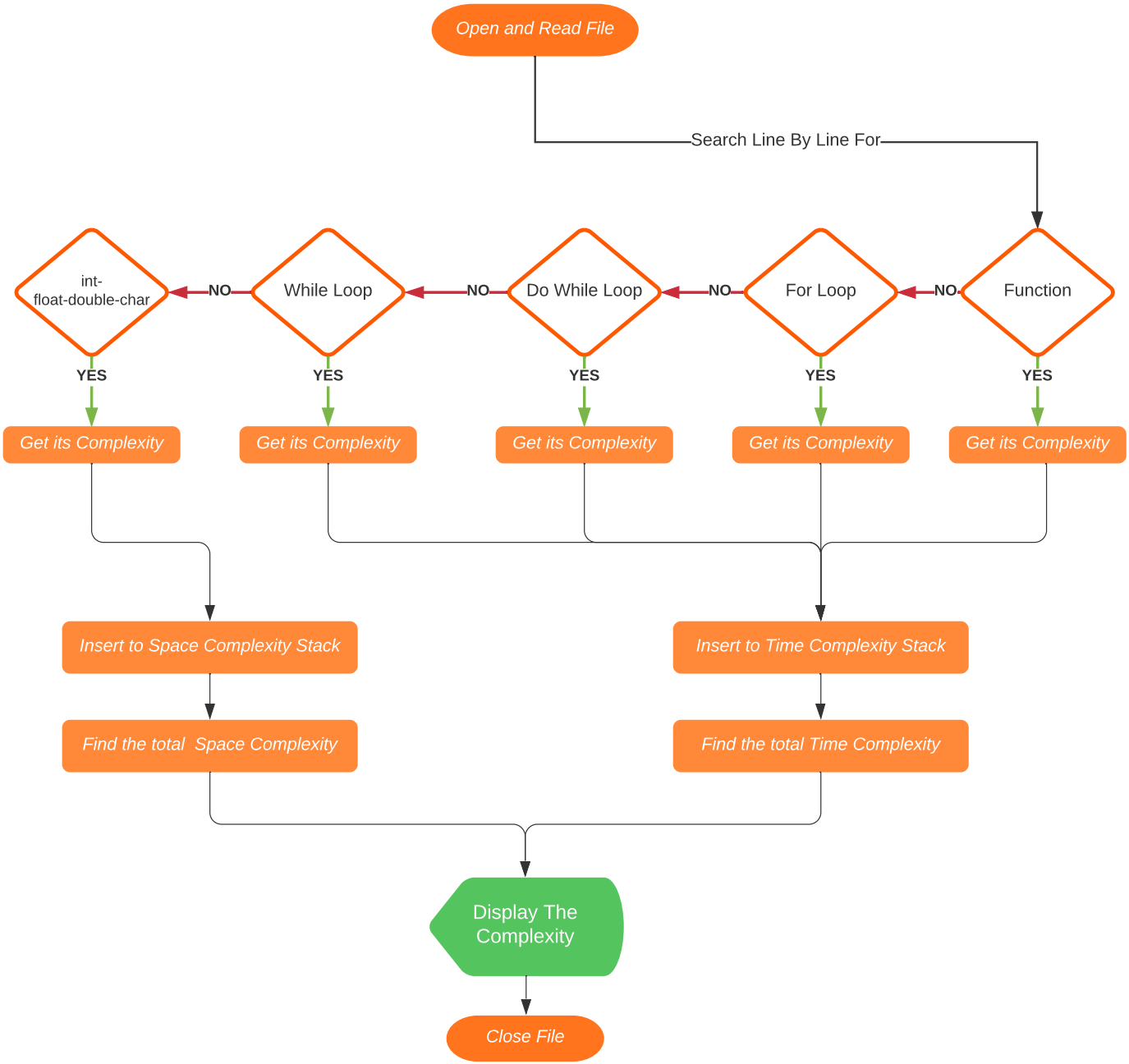
"F:\Computer-Engineering\Second Year\Secon...

```
space Complexity: n

Time Complexity: n


Process returned 0 (0x0)   execution time : 0.122 s
Press any key to continue.
```

REFERENCES

*https://www.tutorialspoint.com/c_standard_library/c_function_strstr.htm

* https://www.tutorialspoint.com/


Open and Read File
Search Line By Line For
Function
For Loop
Do While Loop
While Loop
int-
float-double-char
NO
NO
NO
NO
YES
YES
YES
YES
YES
Get its Complexity
Get its Complexity
Get its Complexity
Get its Complexity
Get its Complexity
Insert to Space Complexity Stack
Insert to Time Complexity Stack
Find the total Space Complexity
Find the total Time Complexity
Display The Complexity
Close File