BLM209 PROGRAMLAMA LABORATUVARI II
PROJE 1

Barış Kakilli
Kocaeli Üniversitesi
Bilgisayar Mühendisliği
200201012@kocaeli.edu.tr

Muhammed Sina Çimen
Kocaeli Üniversitesi
Bilgisayar Mühendisliği
200201032@kocaeli.edu.tr

I.ÖZET

Bu projede amaç text dosyasından kod parçasını okuyarak ve Big-O notasyonu kullanılarak zaman karmaşıklığını(time complexity) ve yer(hafıza) karmaşıklığını(space complexity) tespit etmek, T(n) süre hesabını yapmaktır.

II. GİRİŞ

Zaman ve hafıza karmaşıklığı bilgisayar bilimleri ve benzeri bilimlerde istenilen soruya karşılık her zaman istenilen cevaplar en hızlı veya en kesin sonucu verecek algoritma ve yöntemler olmayabilir. Bu durumun nedeni yazılan yöntem ve algoritmanın verimliliği ile ilgilidir. Algoritma ne kadar verimli çalışır ve istenilene ne kadar yakın olursa kodun performansı o kadar iyi olmaktadır. 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 belirtmektedir. İyi bir algoritmadan az yer kaplaması ve az zaman harcaması beklenir. Problemi çözmek için algoritmanın harcadığı zamanın analizi zaman karmaşıklığını, gerekli belleğin analizi ise yer(space) karmaşıklığının hesabını gerektirir. Hesaplanan karmaşıklıkları analiz etmek ve bunları temsil etmek için, Asimptotik Notasyon kullanılmaktadır.

Big Oh Notasyonu-O(n): Bir algoritmanın çalışma zamanının veya yerinin üst sınırını temsil eder. Big O Notation'ın rolü, bir algoritmanın yürütülmesi için alabileceği en uzun süreyi veya yeri hesaplamaktır, yani bir algoritmanın en kötü durumunu hesaplamak için kullanılır. Aşağıda kullanılan Bazı Big O notasyon gösterimleri yer almaktadır.

Sabitler => O(d1)

Logaritmik => O(logn)

Lineer => O(n)

Loglineer => O(n logn)

Üstel => O(n^a)

```c
#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;

}
```

Şekil 1: Örnek kod parçası

Yukarıdaki örnekte yukarıda anlattıklarımızı örneklememiz gerekirse
zaman karmaşıklığı=O(n2)
yer karmaşıklığı= (4n^2+16 O(n2))
T(n)= 2n^2

III. YÖNTEM

Program C programlama dilinde yazılmış olup, derleyici olarak “GCC”, IDE olarak “CodeBlocks” kullanılmıştır. Programın çalışabilirliği “Windows ˇ 10 64 bit” işletim sisteminde test edilmiştir. Proje bir ekip tarafından geliştirildiğinden dolayı işlevlerin birleşimi, kodun takibi gibi amaçlarla “Github” versiyon kontrol sistemi kullanılmıştır.

Program ilk çalıştığında dosyanın boş olup olmadığını kontrol ediyoruz. Dosya boşsa ya da açılırken bir hatayla karşılaşıldıysa bunu kullanıcıya bildiriyoruz.
Dosya kontrolü tamamlandıktan sonra dosyanın içindeki kod hatasız bir şekilde okunduysa bunu analiz etmeye başlıyoruz ve Big-O notasyonu kontrolü yaparken gerçek hayatta kullandığımız mantıklardan yararlanarak cümlecikleri ve işaretleri kontrol etmeye başlıyoruz. ';' ifadesi ile karşılaştığımızda durum kontrolü yapıp karmaşıklığı uygun bir şekilde artırıyoruz.

Ardından for, do, while, return kelimeleri geçiyor mu diye bir sorgu başlatıyoruz. Bu seçeneklerin herhangi biri ile karşılaşılması durumunda her biri için ayrı ayrı oluşturulmuş algoritmalarımızı kullanıyoruz.

For geçiyorsa parantez içindeki duruma göre döngü lineer mi logaritmik mi kontrol ediyoruz.

While geçiyorsa '}' ara 1 önceki kelimeye geç sonra lineer mi logaritmik mi kontrol ediyoruz.

Do geçiyorsa while kelimesini arıyoruz ve 1 önceki kelimeye geçip ardından lineer mi logaritmik mi hesaplıyoruz.

Return geçiyorsa return bir fonksiyon çağırıyor mu yani recursive bir kod mu diye kontrol ediyoruz. Çağırıyorsa lineer mi logaritmik mi diye bir sorgu kullanıyoruz.

Tüm bu işlemlerden sonra da en önemli kontrollerden birini yapıyoruz: Döngünün toplam kaç adet iç içe olduğunu görmek için test işlemi uyguluyoruz.

Hafıza analizi yaparken de farklı farklı algoritmalar kullanıyoruz. Int, char, double, float kelimeler geçiyor mu kontrolü yapıyoruz.

Eğer geçiyorsa bir sonraki kelimelerde '(' geçiyor mu bakıyoruz(bu kontrolu int main için yapıyoruz) ve '(' geçmiyorsa ',' geçiyor mu bakıyoruz eğer geçiyorsa karmaşıklığı veri tipinin kapladığı alan kadar arttırıp dosyayı okumaya devam ediyoruz.

'[' geçiyorsa içindeki değeri ve veri tipinin kapladığı alanı çarparak karmaşıklığa kaydediyoruz.

Eğer ';' geçiyorsa karmaşıklığı veri tipinin kapladığı alan kadar arttırıp karmaşıklık kontrolünü bu noktada noktalıyoruz ve ardından bizden istenen tüm isterleri sırayla yazdırıyoruz.

Bu projenin sınırsız bir genişlik ve uyumlulukta tanımlamak teknik açıdan mümkün olmadığından dolayı okuyacağımız kodun bazı kurallar çerçevesinde düzenlenmiş olması gerekiyor bu kurallar şöyle:

1-for do ve while'dan sonra parantez için bir boşluk olmak zorunda.

2-do while ve while için arttırma azaltma çarpma işlemleri en sonda(kapalı süslü parantezden 1 önce) olmak zorunda.

3-fonksiyonlar için return kısmında parantez içi bitişik yazılmalı. örnek: return n * factorial(n-1);

4- int içinde birden fazla değer ',' ile ayrılcaksa boşluk bıraılmalı. örnek: int i, a, b;

IV. DENEYSEL SONUÇLAR

```
#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;

}
```

Şekil 2: Kodun dosyadan okunması

```
zaman karmasikligi: O(n^2)
Programin calismasi icin gecen zaman: T( 6n^0 + 2n^2 +)

yer karmasikligi: 16  + nn*4
```

Şekil 3: Karmaşıklık ve zaman analizi

V. SONUÇ

Projede bizden istenen tüm isterler eksiksiz bir şekilde başarıyla tamamlandı. Fakat bu projeyi sınırsız bir genişlik ve uyumlulukta tanımlamak teknik açıdan mümkün olmadığı için belirli kural ve kaideler çerçevesinde sorunsuz sonuç üretmektedir. Bu kurallar yöntem kısmında açılanmıştır.

VI. YALANCI KOD

1-dosya boş mu dolu mu kontrol et

2-dosyanın sonuna kadar döngü başlangıcı

3-dosyadan kelime oku

4-kelime içerisinde ';' karakteri geçiyorsa o anki döngü durumuna göre karmaşıklığı arttır.

5-for, do, while, return kelimeleri geçiyor mu kontrol et

5a-for geçiyorsa parantez içindeki duruma göre döngü lineer mi logaritmik mi hesapla

5b-while geçiyorsa '}' ara 1 önceki kelimeye geç sonra lineer mi logaritmik mi hesapla

5c-do geçiyorsa while ara 1 önceki kelimeye geç sonra lineer mi logaritmik mi hesapla

5d-return geçiyorsa return bir fonksiyon çağırıyor mu bak çağırıyorsa lineer mi logaritmik mi hesapla

6-döngünün toplam kaç adet iç içe olduğunu görmek için test işlemi uygula

7-int, char, double, float kelimeleri geçiyor mu kontrol et

7a-eğer geçiyorsa bir sonraki kelimelerde '(' geçiyor mu bak(bu kontrol int main için)

7b-eğer '(' geçmiyorsa ',' geçiyor mu bak eğer geçiyorsa karmaşıklığı veri tipinin kapladığı alan kadar arttır

7c-eğer '[' geçiyorsa içindeki değeri ve veri tipinin kapladığı alanı çarparak karmaşıklığa kaydet

7d-eğer ';' geçiyorsa karmaşıklığı veri tipinin kapladığı alan kadar arttır

8-kaydedilen zaman karmaşıklığından O(n)i yaz

9-T(n) zaman karmaşıklığını yaz

10-kaplanan alanı yaz

VII. KAYNAKÇA

1) https://www.tutorialspoint.com/c_standard_library/string_h.htm

2) https://slaystudy.com/c-program-to-search-a-word-in-a-given-file-and-display-all-its-position/

3) https://bilgisayarnot.blogspot.com/2020/05/algoritma-zaman-hafza-karmasiklik.html

4) https://ibrahimkaya66.wordpress.com/2013/12/30/10-algoritma-analizi-algoritmalarda-karmasiklik-ve-zaman-karmasikligi/comment-page-1/

5) https://www.javatpoint.com/big-o-notation-in-c

6) https://stackoverflow.com/