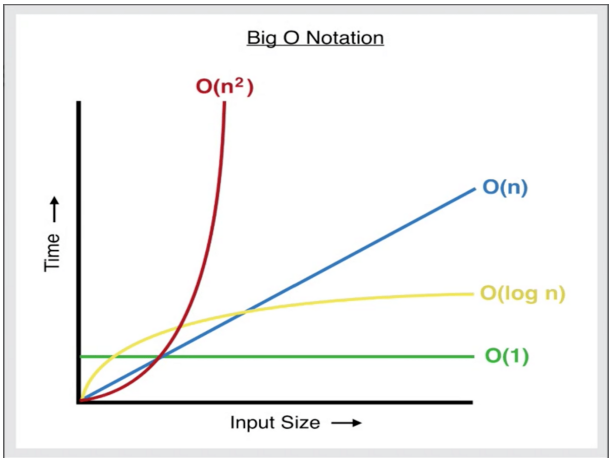
PROLAB2 PROJE-1
Time and Place Complexity Calculation

1st Melih Selami ÜRKMEZ
Kocaeli Üniversitesi
Bilgisayar Mühendisliği
Kocaeli,Türkiye
200202010@kocaeli.edu.tr

***Abstract*—Projede özet olarak asistanlar tarafından verilen kod bloklarının zaman karmaşıklıklarını, yer karmaşıklıklarını,kodun çalıştığı süre boyunca geçen zamanı, ve dosya içeriğinin okunması isterlerinin gerçekleştirilmesi istenmiştir.Zaman ve yer karmaşıklıkları ifadesinin Big-O notasyonu ile gösterilmesi istenmiştir.**

I. GİRİŞ

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.



II. TEMEL BİLGİLER

Program C dilinde geliştirilmiş olup,tümleşik geliştirme programı olarak "CodeBlocks" kullanılmıştır.

III. YÖNTEM

- void dosyadanoku()
  Yukarıdaki fonksiyon verilen input dosyasındaki satırları teker teker okuyarak yerkarmasikligihesapla() fonksiyonuna gönderir. Daha sonra dosya okuma işlemi bittikten sonra dosya kendini kapatır.

- int recursivekontrolet()
  Yukarıdaki fonksiyon, içerisine yollanan satırda bir veri tipi tanımlaması var ise ve aynı satırda açık veya kapalı parantez bulunduruyorsa return 1 ifadesini döndürerek bunun bir fonksiyon olduğunu söyler.

- int forkontrolet()
  Yukarıdaki fonksiyon içerisine yollanan satırda for kelimesinin olup olmadığını kontrol eder.

- int dowhilekontrolet()
  Yukarıdaki fonksiyon içerisine yollanan satırda do kelimesinin olup olmadığını kontrol eden fonksiyondur.

- int forartiskontrolet()
  Yukarıdaki fonksiyon içerisine yollanan for satırındaki artışı kontrol eder.Açıklayacak olursak satır içerisinde + veya - bulunuyorsa bunu 1 olarak , içerisinde * ya da / bulunuyorsa bunu 0 olarak döner.

- int parantezkontrolet()
  Yukarıdaki fonksiyon içerisine yollanan satırda kapanma parantezini görürse 1 döndüren bir fonksiyondur.

- int whilekontrolet()
  Yukarıdaki fonksiyon içerisine yollanan satırda while kelimesinin olup olmadığını kontrol eder.Böylece while zaman ve yer karmaşıklığını hesaplarken elimizde daha temiz bir kod oluşur.

- void gecenzamanihesapla()
  Proje isterleri arasında input dosyasında bulunan kod bloğunun çalışma süresi istenmektedir. Ve kodun kopyala yapıştır yapılarak süre hesaplanmasının yasak olduğu söylenmiştir.
  Bunun üzerine ben de bu fonksiyon ile kod bloğunun çalışma süresini hesaplıyorum.Yaptığı işlem ilk olarak .c uzantılı bir c dosyasını yazmak için açmaktır.Daha sonra input dosyadasındaki satırları teker teker okur.Fakat int main gördükten hemen sonra time t değişkenlerini oluşturur ve süreyi başlatır. Daha sonra return görene kadar devam eder.Return görünce ilk başta süreyi sonlandırır.Başlangıç ve bitiş süresinin arasındaki farkı tanımlar.Ekrana yazdırır ve en son olarak return ifadesini yazarak .c uzantılı dosyayı kapatır.
  Sürenin görünebilmesi için oluşturulan .c uzantılı dosyanın derlenip çalıştırılması gerekir.Direkt input içerisindeki işlemi yaparak ekrana geçen süreyi yazdırır.

- void yerkarmasikligihesapla()
  Bu fonksiyon verilen kod bloğundaki yapıların yer karmaşıklıklarını hesaplayarak hem normal halini hem de Big-O notasyonu ile gösterimini sağlar.İçerisindeki işlemlere gelecek olursak;
  Ilk olarak satırda int,float,double veya char var mı diye kontrol eder.Eğer bunlardan herhangi biri varsa içerisinde ek olarak dizi parantezinin olup olmadığını kontrol eder.Eğer içerisinde dizi parantezi var ise dizi parantezi içerisindeki yapıları alır. Bunları birbiri ile çarpılmış halinde diziye yazar.Başına da hangi veri tipi ise onun boyutunu yazarak yer karmaşıklığını hesaplar.Eğer içerisinde dizi parantezi yok ise;
  Satır içerisinde bulunan virgül sayısını sayar.Virgül sayısının 1 fazlası kadar bulunan veri tipinin bayt karşılığı ile çarparak toplam değişkenine atar.
  Böylece en sonda tüm bunları ekrana yazdırarak yer karmaşıklığının değerini döndürür.

- void zamankarmasikligihesapla()
  Yukarıdaki fonksiyon input dosyasındaki kod bloğunun zaman karmaşıklığını hesaplama işini görür.
  İşlemlerine gelecek olursak;
  İlk başta input dosyasındaki satırları teker teker okuyarak iki boyutlu bir diziye atar.Çünkü satırların üzerinde dolaşmamızın daha rahat olması için dosyayı okuyarak bir diziye atma işlemi yapılır.Daha sonra dizide bulunan satır sayısı kadar dizide dolaşılmaya başlanır.Eğer satır for,while,do while veya recursive fonksiyon kontrollerinden birine girerse içlerinde işlemler yapılarak zaman karmaşıklığı hesaplanır.
  Eğer for kontrolüne girerse;
  Açık ve kapalı parantez sayısı sayılır.Eğer açık ve kapalı parantez birbirine eşitse iç içe olan for sayısı for sayılarının bulunduğu diziye atılarak tüm değerler sıfırlanır. Bu işlem tüm input dosyayı okunana kadar devam eder.
  En sonda tüm okuma işlemi bittikten sonra for sayılarının bulunduğu dizideki en yüksek for sayısı alınır. Onun indexi bulunur.Daha sonra döngü artışlarının bulunduğu matristeki o indexe gidilerek zaman karmaşıklığı ekrana yazdırılır.
  Eğer dowhile kontrolüne girerse;
  Do while döngüsü arasında ++,–,*=,/= ifadeleri aranır.Bu ifadelere göre ekrana zaman karmaşıklığı yazdırılır.
  Eğer while kontrolüne girerse;
  Tekrardan while blokları arasında ++,–,*=,/= ifadeleri aranır.Eğer bu ifadelere denk gelirse gelen ifadeye göre ekrana zaman karmaşıklığı yazdırılır.
  Eğer recursive kontrolüne girerse;
  Parantez bulunduran return ifadesini arar.Parantezden sonra +,-,*,/ olma durumuna göre zaman karmaşıklığını ekrana yazdırır.

- int main()
  Bu ana fonksiyon içerisinde kullanıcının rahatça kullanabileceği bir arayüz tasarlanmıştır.4 adet seçenek kullanıcıya sunulmaktadır.Daha sonra bu istekler kullanıcı sonlandırıncaya kadar kullanıcıya soracak şekilde tasarlanmıştır.Seçme işlemi için switch-case yapısı kullanılmıştır.

IV. KABAKOD

1)Kullanıcıya dosyaların okunduğuna dair bilgi ver.

2)Kullanıcının okunan kod üzerinde yapmak istediği işlemi al.

3)Kullanıcı eğer 1. işlemi seçmiş ise güzel görünüm için ilk başta ekrana 'Time Complexity' yazdır. ve daha sonra yukarıdaki anlatılan zamankarmasikliginihesapla() fonksiyonunu çağır ve ekrana yazdır.

4)Kullanıcı eğer 2. işlemi seçmiş ise güze görünüm için ilk başta ekrana 'Memory Complexity' yazdır ve daha sonra yukarıdaki anlatılan yerkarmasikliginihesapla() fonksiyonunu çağırarak yer karmaşıklığını hesapla ve ekrana yazdır.

5)Kullanıcı eğer 3. işlemi seçmiş ise dosyadaki satırları .c uzantılı bir dosyaya yazdır.int main gördükten sonra time değişkenlerini tanımla.Time 'ı başlat. Daha sonra return görene kadar okuduğun satırları yazmaya devam et.Return görünce ilk başta time 'ı sonlandır.Arasındaki farkı hesaplayarak ekrana yazdır. Daha sonra returnu dosyaya yazdır.

7)Kullanıcı eğer 4. işlemi seçmiş ise programdan return ile çıkış yap.

8)Kullanıcı bunlardan farklı bir işlem seçmiş ise böyle bir seçenek tanımlanmadı şeklinde bir hata mesajı döndür.

V. DENEYSEL SONUÇLAR

```
******Time and Memory Complexity Calculating******
* 1)Calculate to Time Complexity               *
* 2)Calculate to Memory Complexity             *
* 3)Calculate the Elapsed Time                 *
* 4)Exit                                       *
************************************************
                 Melih Selami Urkmez 200202010

Please choose your option:1
This code's time complexity:n*n*
Please choose your option:2
This code's memory complexity: *4*n*n*+*16
BigO Memory Complexity:n*n*
Please choose your option:3
A .c file has been created for the elapsed time. See the elapsed time to compile!
File name's:gecenzaman.c
Please choose your option:4
Exited the program! Good Bye!
Process returned 0 (0x0)   execution time : 12.474 s
Press any key to continue.
```

```
******Time and Memory Complexity Calculating******
* 1)Calculate to Time Complexity               *
* 2)Calculate to Memory Complexity             *
* 3)Calculate the Elapsed Time                 *
* 4)Exit                                       *
************************************************
                 Melih Selami Urkmez 200202010

Please choose your option:1
This code's time complexity:n
Please choose your option:2
This code's memory complexity: *4*n*+*12
BigO Memory Complexity:n*
Please choose your option:3
A .c file has been created for the elapsed time. See the elapsed time to compile!
File name's:gecenzaman.c
Please choose your option:4
Exited the program! Good Bye!
Process returned 0 (0x0)   execution time : 6.349 s
Press any key to continue.
```

```
******Time and Memory Complexity Calculating******
* 1)Calculate to Time Complexity               *
* 2)Calculate to Memory Complexity             *
* 3)Calculate the Elapsed Time                 *
* 4)Exit                                       *
************************************************
                 Melih Selami Urkmez 200202010

Please choose your option:1
This code's time complexity:n
Please choose your option:2
This code's memory complexity: *8
BigO Memory Complexity:
Please choose your option:3
A .c file has been created for the elapsed time. See the elapsed time to compile!
File name's:gecenzaman.c
Please choose your option:4
Exited the program! Good Bye!
Process returned 0 (0x0)   execution time : 8.488 s
Press any key to continue.
```

VI. SONUÇ

Bu projeden çıkardığım sonucu özetleyecek olursam ilk başta yazdığımız kod için verimlilik ve hızın ne kadar önemli olduğunu araştırdım.Kodumuzun iyiliği aslında sadece istenen probleme hizmet etmesi değil ayrıca hızlı,verimli az maliyetli olması da bu özelliklerin arasında yer almalı olması gerektiği öğrendim.Bunun için araştırmalarda bulundum.Ve bu verilen projenin tam olarak yapılamayacağı kanaatine vardım.Çünkü bunu C dilinde yapmak işleri çok zorlaştırıyor.Herhangi bir makine öğrenme aracılığı işimizi daha çok kolaylaştırabileceğini düşünüyorum.

Bu proje üstünde çalışırken kendime kattığım ve tekrar ettiğim bilgileri paylaşmak istiyorum;

1)Dosyadan veri okumak,veri yazmak gibi dosya işlemlerini tekrar ettim.

2)Daha sonra .c uzantılı dosyaya nasıl veri yazılır ve bunu kod içerisinde derlememizin bir yolu olup olmadığını araştırdım.

3)String fonksiyonlarını tekrar ettim.İçerisinden isalpha gibi birçok metodu kullandım.

4)Sıralama algoritmalarını tekrar ettim.

5)Queue yapısı tekrar ettim.

6)Birçok string üzerinde fonksiyonlar yazarak kodumun daha temiz olmasını sağlamaya çalıştım.

VII. KAYNAKÇA

1)https://medium.com/kodcular/nedir-bu-big-o-notation-b8b9f1416d30

2)https://tex.stackexchange.com/questions/25895/adding-a-new-page

3)https://www.youtube.com/watch?v=seRhO4to–Y

4)http://www.oguzhantas.com/c-programlama/379-standart-c-ile-dosyadan-veri-okuma-ve-yazma.html

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

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

7)https://medium.com/dataseries/how-to-calculate-time-complexity-with-big-o-notation-9afe33aa4c46

8)https://shunnarski.github.io/BigO.html

9)https://stackoverflow.com/questions/3255/big-o-how-do-you-calculate-approximate-it

10)https://www.inoutcode.com/concepts/big-o/