PROGRAMLAMA LABORATUVARI PROJE 2

Edanur Çövüt-190201035

Fedai Engin Can Yılmaz-200201042

20 Mart 2022

1 Özet

Bu dökuman Programlama Laboratuvarı 2 dersi 1. Projesi için hazırlanmış. Dökumanda projenin tanımı, çözüme yönelik yapılan araştırmalar, kullanılan yöntemler, proje hazırlarken kullanırken kullanılan geliştirme ortamı ve kod bilgisi gibi programın oluşumunu açıklayan başlıklar bulunmaktadır. En sonda da kaynakça kısmı bulunmaktadır.

2 Proje Tanımı

Bu proje ile, C dilinde kullanılan kodların zaman ve yer karmaşıklığını hesaplaması amaçlanmaktadır.

Zaman ve Hafıza Karmaşıklığı

Bilgisayar bilimleri 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(1)
Logaritmik = O(logn)
Lineer = O(n)
Loglineer = O(n logn)
Üstel = O(n*a) || O(a*n)

Örnek OLARAK;

```
void printAllNumbersThenAllPairSums(int arr[], int size)
{
   for (int i = 0; i < size; i++)
   {
      printf("%d\n", arr[i]);
   }
   for (int i = 0; i < size; i++)
    {
```

```
        for (int j = 0; j < size; j++)
        {
            printf("%d\n", arr[i] + arr[j]);
        }
    }
}
```

Zaman karmaşıklığı: O(n+n2) = O(n2)

Projede bizden istenenler aşağıdaki gibidir:

A) Dosya içerisinden kodu okunması ve Dosyanın içeriğinin kontrol edilmesi

B) Dosyadan okunan kod Big O notasyonuna göre Zaman karmaşıklığının hesaplanması

C) Dosyadan okunan kod Big O notasyonuna göre yer (Hafıza) karmaşıklığının hesaplanması

D)Dosyadan okunan kod çalıştırıldığında geçen süresin hesaplaması

3 Araştırmalar ve Yöntem

Bizden girilen verinin zaman ve yer karmaşıklığının çözülmesi istendi. Bu fonksiyonlar for,while, rekürsif vb. olabilir. Bunların hepsinde çalışacak bir proje yaptık. Öncelikle Big-O notasyonunu(algoritma analizi) araştırmak önceliğimiz oldu.

Algoritmanın en kötü durum analizini yapmak için kullanılan notasyondur. Matematiksel olarak şöyle tanımlanır: f(x) ve g(x) reel sayılarda tanımlı iki fonksiyon olmak üzere, x büyüktür k olacak şekilde bir k vardır öyle ki, |f(x)| küçüktür C*|g(x)| dir ve şeklinde gösterilir. Yer karmaşıklığında edindiğimiz bilgiler: Yürütme zamanı, ‘n’ boyutlu bir problemin algoritmasını çalıştırmak için gerekli olan zamana denir. Başka bir ifadeyle karşılaştırma, döngü çevirimi, aritmetik işlemler gibi algoritmanın işlevini yerine getirmesi için temel kabul edilen işlemlerin kaç kere yürütüldüğünü veren bir bağıntıdır. Bir kodlama problemi için mümkün olan birkaç algoritma olabilir ve bunların arasından verimli algoritmalara karar vermemiz gerekir. En pratik senaryoda, girdi boyutumuz çok büyük, yani milyonlarca veya milyarlarca aralığında olacaktır. Bu nedenle, temel fikir, her bir algoritma için (yukarıda hesaplandığı gibi) tam olarak en kötü durum çalışma süresini karşılaştırmak ve verimliliğe karar vermek olacaktır. Ancak bu, birkaç satır kod içeren algoritmalar için matematiksel olarak karmaşık ve zor olacaktır.

Girdi boyutunu artırırsak çalışma süremiz de artar. Bu büyüme, çoğunlukla, çalışma süresi işlevindeki daha yüksek dereceli terime bağlıdır. Örneğin: en kötü durumdaki çalışma süresinin ikinci dereceden bir fonksiyon $an^2 + bn + c$ olduğunu varsayalım. Büyük bir n değeri için, düşük dereceli terimler, yüksek dereceli terimlere kıyasla önemsizdir, yani, $an^2 >> bn + c$. Daha iyi anlamak için, en kötü durum çalıştırma süresinin

$3n^2 + 2n + 1$

olduğunu varsayalım, burada

a = 2, b = 2, c = 1.

Örneğin n elemanlı bir küme için yürütme zamanı T(n) şeklinde gösterilir.

Ayrıca algoritmadaki eleman sayısı fazlalaştığında yürütme zamanı, zaman karmaşıklığı olarak da adlandırılır. Aslında yapılan iş bakımından yürütme zamanı ile zaman karmaşıklığı benzer olsa da zaman karmaşıklığını yürütme zamanından ayıran çok önemli ayrıntılar var.

Çalışma süresi hesabında dikkat ettiğimiz temel hesap birimlerine göz atalım:

Programlama dilindeki deyimler

- Döngü sayıları
- Toplam işlem sayısı
- Dosyaya erişim sayısı
- Atama sayısı

Yazdığımız kodun çalışabilirliğine dair 2 örnek verecek olursak.

Dosya uygun olup kod çalıştığında:

```
PS C:\Users\engin> cd "d:\c_vscode_test"
PS D:\c_vscode_test> & .\"prolab2sonhali.exe"

Okunan kod parcacigi:
#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++;
    }while(i<=n);
    return 0;
}
 //zaman karmasikligi=O(n).
 //yer karmasikligi=(4n+12 O(n)
---------------------------------


Toplam Zaman Karmasikligi:   O(n)
Toplam yer karmasikligi  :   4n + 8
Gecen sure               :   0.009000 ms
```

Dosya bulunamadığında;

```
PS D:\c_vscode_test> cd "d:\c_vscode_test"
PS D:\c_vscode_test> & .\"prolab2sonhali.exe"
Dosya acilamadi.
```

Başka bir örnek verildiğinde:

```
PS D:\c_vscode_test> cd "d:\c_vscode_test"
PS D:\c_vscode_test> & .\"prolab2sonhali.exe"

Okunan kod parcacigi:
#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;

}
 //zaman karmasikligi=O(n).
 //yer karmasikligi=(n+8 O(n)
---------------------------------


Toplam Zaman Karmasikligi:   O(n)
Toplam yer karmasikligi  :   16
Gecen sure               :   0.003000 ms
```

4 Geliştirme Ortamı

Projeyi Windows sistemde, CodeBlocks üzerinde geliştirip yine CodeBlocks kullanarak derledik.

5 Kod Bilgisi

5.0.1 Akış Diyagramı

Akış diyagramı klasör içindedir.

5.0.2 Algoritma

5.0.3 İstatistik

Program kodu 670 satırlık tek dosyadan oluşmaktadır. Kod düzenini sağlamak için yaklaşık 45 boş satır ve 98 yorum satırı kullanılmıştır.Kullandığımız kütüphaneler ve ne için kullandığımız kabaca aşağıdaki gibidir:

<stdio.h>

Çıktı ve girdi almak için

<stdlib.h>

Malloc fonksiyonunu kullanmak için

<time.h>

Başlık dosyasında tarih ve saat işlemleri ile ilgili fonksiyon, makro ve değişken tanımlamaları yer almaktadır.

<ctype.h>

Bir byte değerin alfabede bulunan bir değer ve bir rakam olup olmadığını kontrol eder.

5.0.4 Programın Derlenmesi

Programın kaynak kodu tek dosyadan oluşmaktadır. Bu dosyayı CodeBlocks, C++ veya Visual Studio Code ile derleyebilirsiniz.

6 Kaynakça

https://ichi.pro/tr/veri-yapisi-ve-algoritmalarda-zaman-karmasikligi-analizi-232261939322624

http://www.adau.edu.az/images/ms_kitabxana/23_07_2021_13_26_00_89_561603_Algoritma%20Karma%C5%9F%C4%B1kl%C4%B1%C4%9F%C4%B1%20ve%20B%C3%BCy%C3%BCk%20O%20G%C3%B6sterimi.pdf

https://www.youtube.com/watch?v=3bhBo9YCTpo

https://www.youtube.com/watch?v=XQqjUSOi45o

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

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

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

https://www.youtube.com/watch?v=xRRTRt8MJFg

https://www.youtube.com/watch?v=E6DjPQE4Y_4

https://www.youtube.com/watch?v=OMjIETZUbgo