Programlama Laboratuvarı 2

.

Umut SÜTCÜ 200202038

Ömer ARAN 190202012

ÖZET

Bu doküman programlama laboratuvarı 2 dersinin 1.projesi olan zaman ve yer karmaşıklığının hesaplanma projesi için çözümü açıklamaya yönelik oluşturulmuştur. Bu dökümanda proje tanıtımı, çözüme yönelik yapılan araştırmalar, kullanılan yöntemler, geliştirme ortamı ve kullanılan kaynaklara yer verilmiştir.

PROJENIN TANIMI

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 1

```
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;
```

Yukarıdaki örneğin zaman karmaşıklığı: O(n^2)

Yer (hafıza) karmaşıklığı ise 4n^2 + $16O(n^2)'dir$

Örnek 2

```
int main()


int count=10,i,n=10;

int arr[n];

i=1;

do

arr[i]=i*count;

printf("%d * %d = %d",i,count,i*count);

i++;

while(i¡=n);

return 0;
```

Yukarıdaki örneğin zaman karmaşıklığı=O(n)

Yer (hafıza) karmaşıklığı ise 4n+12 O(n) 'dir

ARAŞTIRMA VE YÖNTEMLER

İlk önceliğimiz “for” ve “while” gibi kelimeleri bulabilmek için string içinde string arama üzerinde çalıştık bunun için “strstr” fonksiyonu ve kendi yazdığımız fonksiyonu kullandık. Daha sonra bunların üstüne kurarak zaman karmaşıklığı ve hafıza karmaşıklığını hesaplamadık. Geliştirme ortamı olarak C compiler ve IDE olan visual studio code ve code blocks kullandık

```
char *strstr (const char *s1, const char *s2);

Parameters:
s1: This is the main string to be examined.
s2: This is the sub-string to be searched in s1 string.
```

Yukarıdaki fotoğrafta strstr fonksiyonunun nasıl kullanılacağına yönelik ufak bir not var.

```c
int main()
{
    // Take any two strings
    char s1[] = "Fun with STL";
    char s2[] = "STL";
    char* p;

    // Find first occurrence of s2 in s1
    p = strstr(s1, s2);

    // Prints the result
    if (p) {
        strcpy(p, "Strings");
        printf("%s", s1);
    } else
        printf("String not found\n");

    return 0;
}
```

Yukarıdaki fotoğrafta da örnek bir kullanımı var

Dosyadan okumayı satır satır yaparak her satırda işlemlerimizi tekrar ettik

```c
FILE *file;
  file = fopen("code.txt", "r");
  if (file == NULL)
  {
      printf("Dosya acilamadi\n");
      exit(1);
  }
  else
  {
      printf("Dosya basariyla acildi\n");
  }

  int a=1;
  char dizi[100];
  do {
     //printf("adim: %d\n",a);
      a++;
      fgets(dizi, 80, file); // fgets() fonksiyonu ile dosyadan satir okuma
      printf(dizi);
      ZamanKarmasikligi(dizi);
      HafizaKarmasikligi(dizi);
  } while (!feof(file));
```

Kodumuzda kullandığımız dosya fonksiyonu

En çok sıkıntı çektiğimiz de nereye kadar aramaya yapacağımızdı örneğin int main()’de de int geçtiği için hafıza karmaşıklığını arttırıyordu biz de bunun için o satırda ”;” bulunup bulunmadığını kontrol ettik bunun için de alttaki bir kontrol mekanizması oluşturduk

```c
if(search_for_word(series,word)==1 && strchr(series,';')!= NULL )
{
    sayma++;

    for(int i= 0; i<80; i++)
    {
        if(series[i]=='[')
        {
            paranthess[sayma]+=1;
            matrixValue=character;
        }


        else if(series[i]==',')
        {
            space_counter+=4;
        }
    }
    if(paranthess[sayma]>=1)
    {
        space_counter-=4;
    }
    space_counter+=4;
}
```

Kodumuzda kullandığımız hafıza karmaşıklığı kontrolü

KOD BILGISI(YALANCI KOD)

Create and open a file

Give for options :

Do

Print Choose an option below

click 1 to see complexity time.

click 2 to see complexity space.

click 3 to see the execution time of code.

click 0 to exit.

Case 1 : run function of time complexity()

Break;

Case 2 : run function of space complexity()

Break;

Case 3 : run function of time execution()

break ;

Case 4 : exit (1) ;

while(user doesn't click to 0);

Read codes.txt with read mode

İf file doesn't find give a warn(dosya açılamadı)and exit to program ;

Else if

continues to program

Check that is there any code in file

İf true

Give a info : (Kod bloğu bulundu.)

Time complexity()

Search while, for, do while from files

Run the algorithm

if find add n to result of big-O notation

if is there nested loops add $n^2 tobig - O$

Write console that time complexity of the code

Print (time complexity is ...)

Space complexity()

Search while, for, do while from files

if find add n to result of big-O notation

Search for variable from files

if find add n to result of big-O notation

Add all byte values of codes

Run the algorithm

if is there series or matrix add necessary things to space complexity

Write console that space complexity of the code

Print (space complexity is ...)

Time execution()

Search while, for, do while from files

Run the algorithm

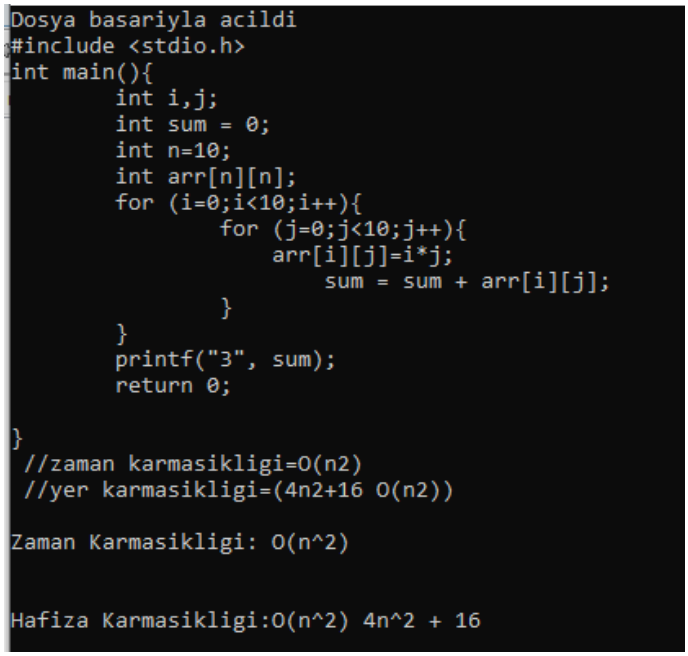
if find add n to result of big-O notation

if is there nested loops add $n^2 tobig - O$

Write console that time complexity of the code

SONUÇ

Sonuç olarak stringler üzerinde arama yapma şekillerini bulup öğrenip c programlama dilinde yazamaya ve ortaya zaman ve hafıza karmaşıklığını bulan bir program çıkarmaya çalıştık. Bilgisayar bilimlerinlerinde mutlaka üzerinde durulması gereken bir proje olduğu sonucuna vardık .Stringler üzerinde işlem için birçok hazır fonksiyon bulunmakta ve bunların birçok kullanım şekli var bunların üzerinde çalışmış olduk.



```
Dosya basariyla acildi
#include <stdio.h>
int main(){
        int i,j;
        int sum = 0;
        int n=10;
        int arr[n][n];
        for (i=0;i<10;i++){
                for (j=0;j<10;j++){
                    arr[i][j]=i*j;
                        sum = sum + arr[i][j];
                }
        }
        printf("3", sum);
        return 0;

}
 //zaman karmasikligi=O(n2)
 //yer karmasikligi=(4n2+16 O(n2))

Zaman Karmasikligi: O(n^2)


Hafiza Karmasikligi:O(n^2) 4n^2 + 16
```

Yukarıdaki fotoğraf kod çıktımızın bir ekran görüntüsü.İlk önce dosyayı okuyup yazdırıyor ve sonra zaman ve hafiza(yer) karmaşıklığını hesaplayarak ekrana bastırıyor.

KAYNAKÇA

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.javatpoint.com/big-o-notation-in-c

https://slideplayer.biz.tr/slide/2658582/
https://www.geeksforgeeks.org/strstr-in-ccpp/