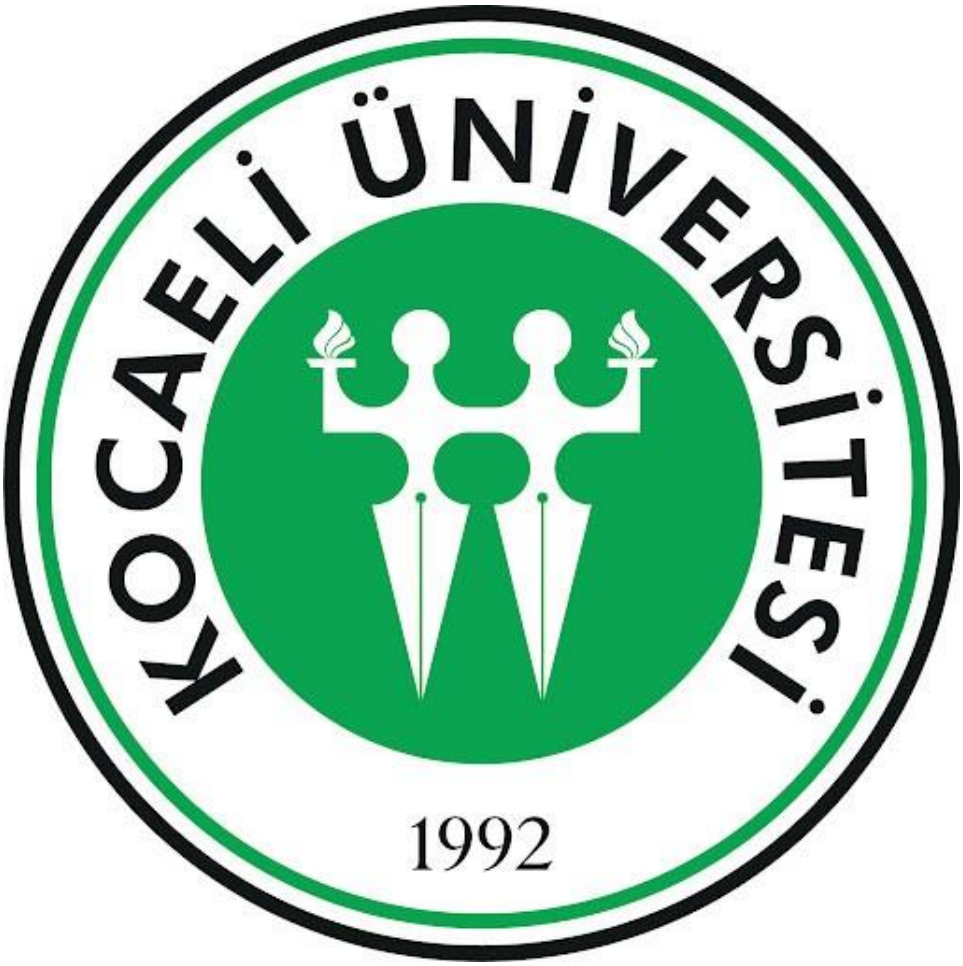
KOCAELİ ÜNİVERSİTESİ
BİLGİSAYAR MÜHENDİSLİĞİ



PROGRAMLAMA LABARATUVARI II
2. PROJE

200201009 - ENİS UZAN
210201106 - YUNUS EMRE YOLCU

YER VE ZAMAN KARMAŞIKLIĞI HESAPLAMA PROGRAMI

Enis Uzan, Yunus Emre Yolcu

Bilgisayar Mühendisliği Bölümü, Kocaeli Üniversitesi, Umuttepe Kampüsü, İzmit/KOCAELİ

uzanenis@outlook.com, yemrewho@outlook.com

Özetçe

Bu projenin amacı, C dilinde kullanılan kodların zaman ve yer karmaşıklığının hesaplaması amaçlanmaktadır.

Bilgisayar bilimleri ve benzeri bilimlerde istenilen soruya karşılık her zaman istenilen cevaplar en hızlı veya en kesin sonucu verecek yöntemler olmayabilir. Her zaman bulunduğumuz durumları göz önünde bulundurarak sonuçlar üretmeyiz. Bir proje için bazen zaman daha önemli olacakken bazen de kesinlik ve bu yönden deharcanacak bellek miktarı daha önem arz etmiş olabilir.

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. Burada çoğu zaman bir takas söz konusu olabiliyor.

Bu projenin asıl amacı da algoritmaların çalışma mantığı çervesinde o algoritmanın zaman ve yer karmaşıklığını çıkarmak ve hesaplanan değerler üzerinden çıkarımlarda bulunmaktır.

1. GİRİŞ

AT&T Bell laboratuvarlarında, Ken Thompson ve Dennis Ritchie tarafından UNIX İşletim Sistemi' ni geliştirebilmek amacıyla B dilinden türetilmiş yapısal bir programlama dilidir.

Geliştirilme tarihi 1972 olmasına rağmen yayılıp yaygınlaşması Brian Kernighan ve Dennis M. Ritchie tarafından yayımlanan "C Programlama Dili" kitabından sonra hızlanmıştır.

Günümüzde neredeyse tüm işletim sistemlerinin (Microsoft Windows, GNU/Linux, *BSD, Minix) yapımında %95' lere varan oranda kullanılmış, hâlen daha sistem, sürücü yazılımı, işletim sistemi modülleri ve hız gereken her yerde kullanılan oldukça yaygın ve sınırları belirsiz oldukça keskin bir dildir.

1.1 Yer Karmaşıklığ Nedir

Zaman karmaşıklığı algoritmanın tamamlanma süresine karşılık gelmektedir. Burada algoritmanın gerçekleştireceği işlemler büyük önem taşır. Belli algoritmalar için bu işlemleri toplam bir sayı olarak belirtebiliriz ve genel terminolojiye göre de N diyelim. Örnek vermek gerekirse eğer elimizde bir sıralama problemi var ise ve sıralayacağımız bir dizi ise sıralanacak eleman sayısı N sayımıza eş değerdir ki bu dizinin boyutudur. Elimizdeki problemi bir graf problemi

olarak düşünürsek de toplam düğüm ve doğruları N ile ifade etmemiz gerekir. Bir algoritmanın çalışma süresi çok fazla etkene bağlıdır. Kullandığı bilgisayarın özelliklerinden kullandığı işletim sistemine arkada çalışan programlara gibi sayılabilecek bir çok faktör buna etki edebilir. Karmaşıklığı hesaplarken bu faktörler göz önüne alınmadan sadece kod üzerinde analiz yapılarak sonuca ulaşılır. Burada asimptotik notasyon (Asymptotic notation) adı verilen bir standart notasyon uygulamaktayız. Bu standart ile bu bağımlılıklardan kurtularak kod üzerinde bir analiz yapabilmekteyiz. Öncelikle bu notasyondaki birkaç terimden bahsetmeliyiz.

-Big O Notasyonu – en kötü durum
-Big Omega Notasyonu (Ω) - en iyi durum
-Big Teta Notasyonu (Θ) – ortalama durum

1.2 Hafıza Karmaşıklığı Nedir

Hafıza karmaşıklığı algoritmanın işlevini yerine getirmesi için kullandığı bellek miktarı olarak söylenebilir. Bunun için öncelikle genel olarak bilinen programlama dilleri için veri yapıları ve onların boyutlarını verelim. Bunların belli programlama dillerine göre değiştiğini unutmamalıdır.

Bool, char, unsigned char, __int8 için **1 byte**
__int16, short, unsigned short **2 byte**
Float, int, long, unsigned int,__int32 **3 byte**
Double, __int64, long double **4 byte**

2. YÖNTEM

Projenin geliştirilmesi için IDE olarak Visual Studio Code kullanılmıştır. Ve C kodlarının derlenmesi için de ''GCC Compiler" kullanılmıştır.

Projenin ilk isterini gerçekleştirmek amacıyla öncelikle ***readFile*** fonksiyonu tanımlanmıştır. Bu fonksiyon dosya adı alarak çalışmaktadır. Dosyamızda bulunan kod bu şekilde sisteme entegre edilmektedir.

Projenin amacına uygun ilerleyebilmek için istenen her karakterin veya karakter dizisinin indisine ihtiyaç duyulmuştur. Bu sebep ile ***findIndex*** adında bir fonksiyon kullanılmıştır. Integer tipindeki fonksiyon bize istediğimiz karakter veya karakter dizisinin indisini vermektedir. Projenin ilerleyişi açısından çok önemli yer tutmaktadır.

Verilerin tutulması amacıyla farklı diziler oluşturulmuştur. Oluşturulan dizilerin isimleri tutacağı karakter veya karakter dizisine özgüdür. Elde edilen indislerle farklı algoritmalar kurulmuştur.

2.1 Zaman Karmaşıklığı Yöntemi

Zaman karmaşıklığını elde etmek için öncelikli olarak ihtiyaç olan durumları elde etmek gerekmektedir. Bunlar için fonksiyonlar ve çeşitli algoritmalar kullanılmıştır. Gerekli olacak şeyler sırasıyla ***for , while, do while*** ve döngüler içerisindeki ***artış operatörleridir.***

Örnek olarak for döngüsü içerisinde 3. Parameter olan artış parametresi için kullanılan algoritma for döngüsünün ikinci noktalı virgülünden ";" for döngüsünün kapatma parantezine ")" kadardır. Ardından izlenecek yol ise basit koşul ifadeleridir. Örnek olarak "++" operatörü için **O(N),** "/=" ifadesi için **O(logN)** ifadeleri yazılmıştır. Ayrıca belirtilen durumlar iç içe döngülerde de control edilmiştir.

Recursive fonksiyonlar için izlenilen algoritma ise öncelikle recursive tespiti ile çalışmaktadır. Recursive tespitinden sonra ise fonksiyonun aldığı parametreye göre işlem görmektedir. Örnek olarak fonksiyon parametresi n ise notasyon **O(N)**'dir parametre n/2 ise notasyon **O(logN)**'dir bu gibi algoritmalar projenin genelinde bulunmaktadır.

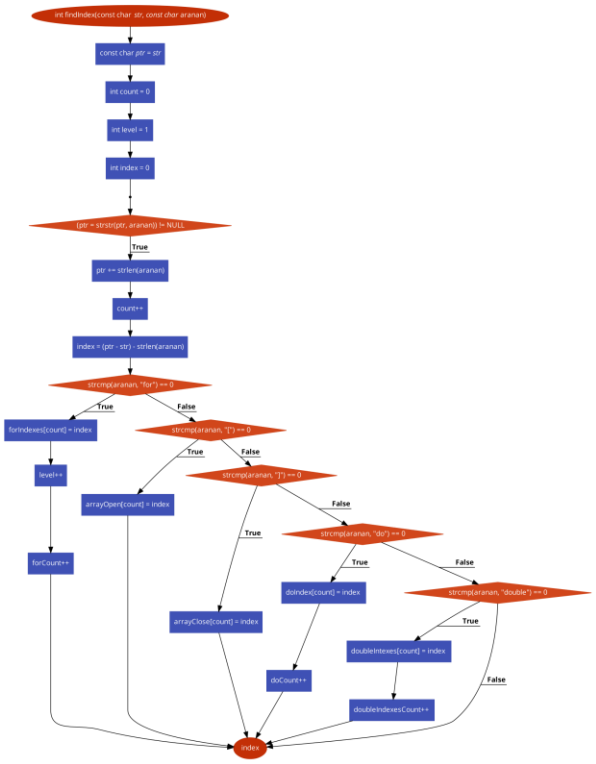
2.2 Yer Karmaşıklığı Yöntemi

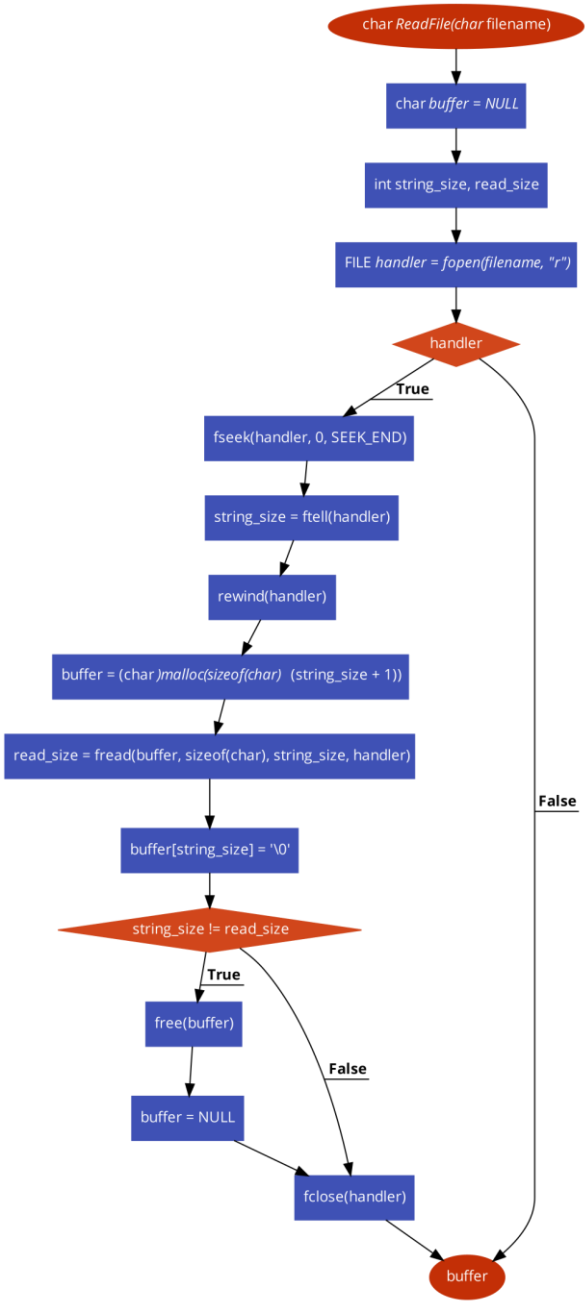
Yer karmaşıklığı algoritmalar için önemli yere sahiptir. C gibi programlaması kullanıcıya bakan dillerde bu çok daha önemli seviyededir. Değişkenin tipini yani bellekte işgal edeceği alanı programı yazan yazılımcı belirlemektedir. Yer karmaşıklığının çıkışı da buna dayanmaktadır. Bir kodun işgal ettiği yer ne kadar az ise sonuç da o kadar iyi olacaktır. Örnek verecek olursak bir

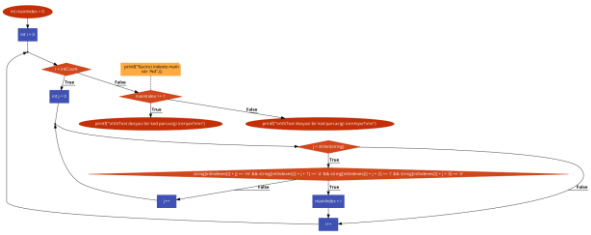
insan topluluğunun evlilik durumlarını char veri tipi ile tutalım ve kişi EVLİ olsun sonuç olarak dört karakterli bir diziye ihtiyacımız oluyor ve bellek hacmi 4 byte oluyor. Bu durumu bool veri tipi ile tuttuğumuzu düşünelim iki ihtimal vardır ya True ya False ve bellekteki yerimiz de 1 byte olacaktır. Görüldüğü gibi belleğimiz 4 kat verimli kullanılmış oldu. Bunun sonucunda anlaşılacağı üzere veri tipi seçimi yer karmaşıklığı için kilit noktada önem göstermektedir.

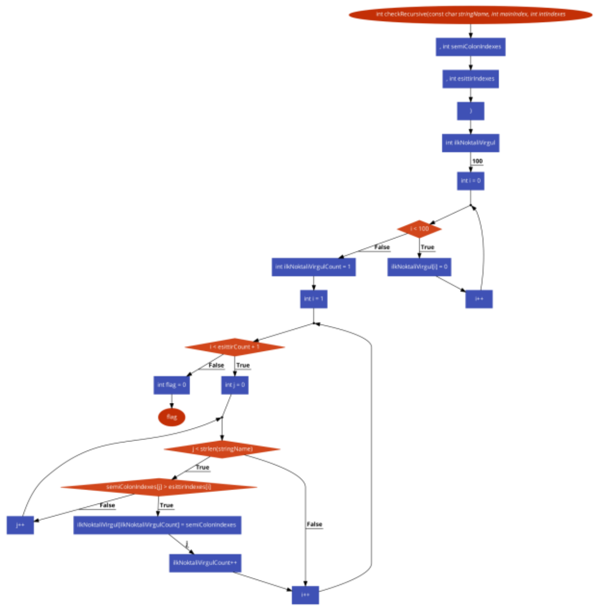
Yer karmaşıklığı hesabında daha önce de sözü geçen findIndex fonksiyonu aktif rol oynamıştır sırasıyla int, float, char vb. Değişken tiplerini saptayarak kendi boyutları kadar çarpılmış ve bellekte kapladığı yer hesaplanmıştır. Ayrıca dizi gibi büyük veri tipleri de bu hesapta önemli yer tutmaktadır.

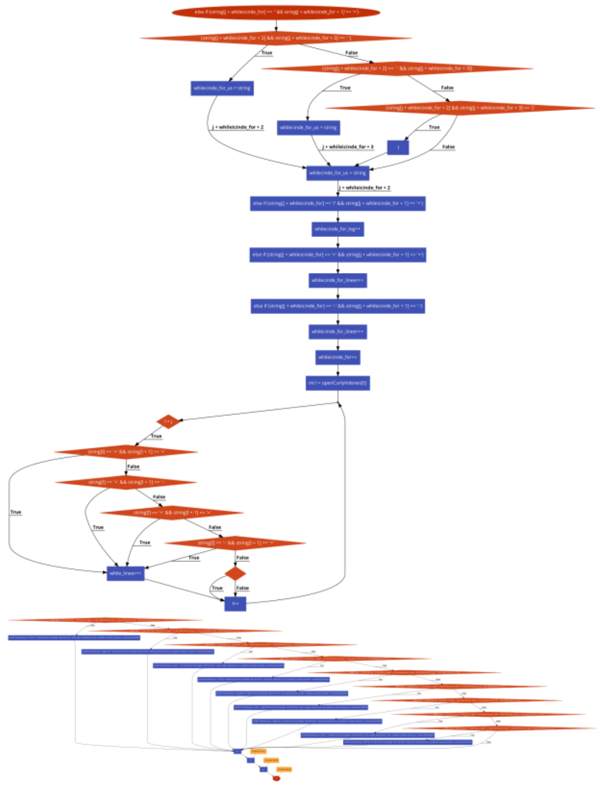
3. DİYAGRAMLAR

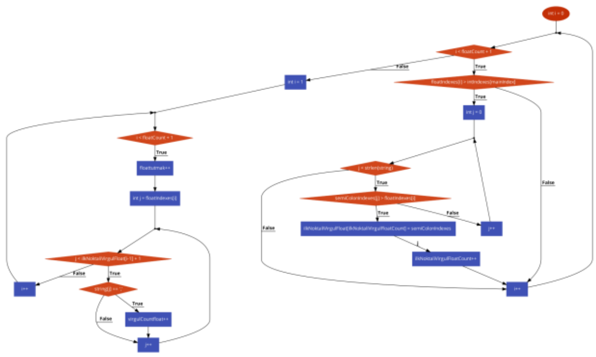


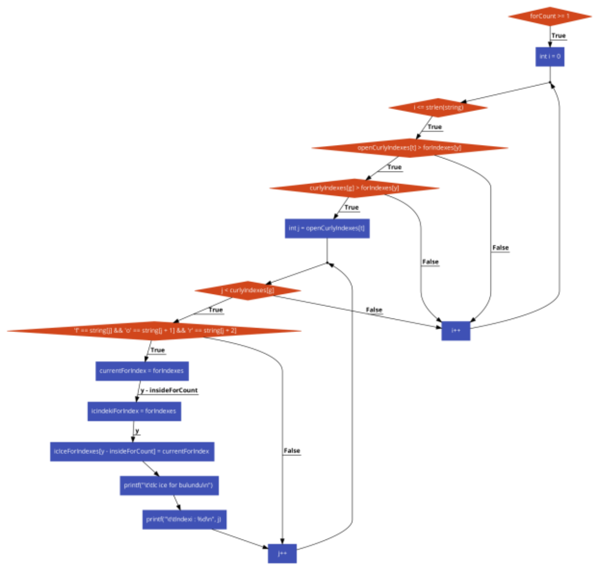


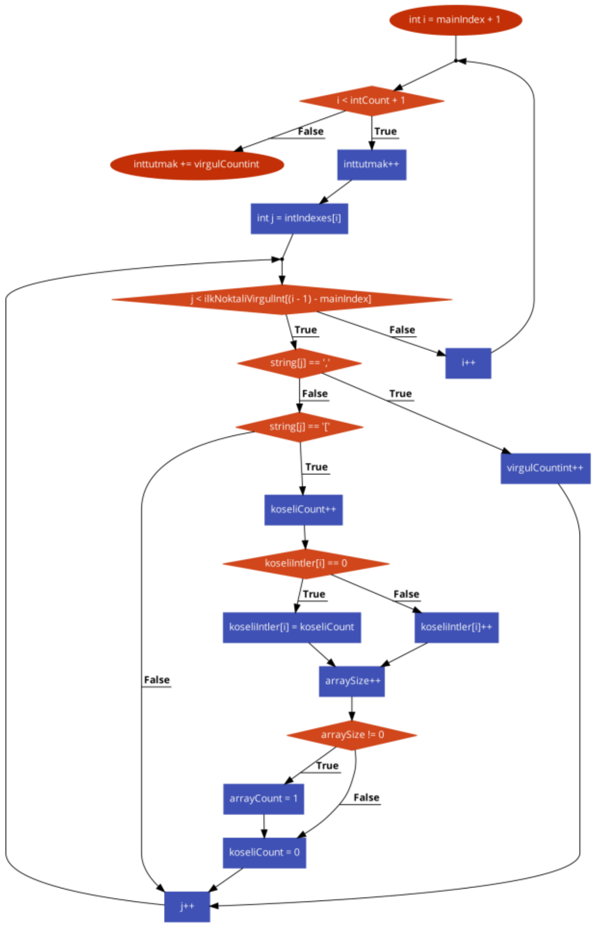



int checkRecursive(const char stringName, int mainIndex, int intIndexes
, int semiColonIndexes
, int esittirIndexes
)
int ilkNoktaliVirgul
100
int i = 0
i < 100
False
True
int ilkNoktaliVirgulCount = 1
ilkNoktaliVirgul[i] = 0
i++
int i = 1
i < esittirCount + 1
False
True
int flag = 0
int j = 0
flag
j < strlen(stringName)
True
semiColonIndexes[j] > esittirIndexes[i]
False
True
j++
ilkNoktaliVirgul[ilkNoktaliVirgulCount] = semiColonIndexes
j
ilkNoktaliVirgulCount++
False
i++

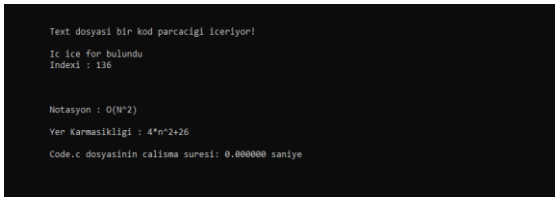


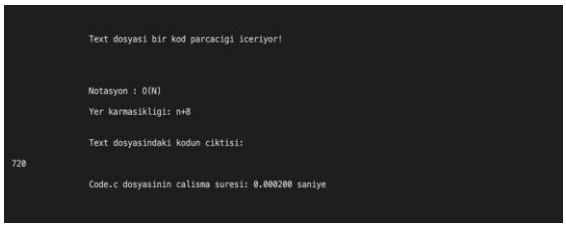


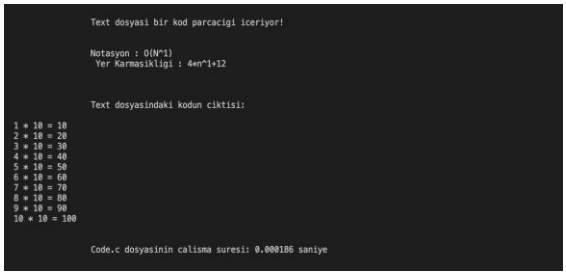

forCount >= 1
True
int i = 0
i <= strlen(string)
True
openCurlyIndexes[t] > forIndexes[y]
True
curlyIndexes[g] > forIndexes[y]
True
int j = openCurlyIndexes[t]
False
False
j < curlyIndexes[g]
True
False
i++
'f' == string[j] && 'o' == string[j + 1] && 'r' == string[j + 2]
True
currentForIndex = forIndexes
y - insideForCount
icindekiForIndex = forIndexes
y
icIceForIndexes[y - insideForCount] = currentForIndex
printf("\t\tIc ice for bulundu\n")
printf("\t\tIndexi : %d\n", j)
False
j++


int i = mainIndex + 1
i < intCount + 1
False
True
inttutmak += virgulCountint
inttutmak++
int j = intIndexes[i]
j < ilkNoktaliVirgulInt[(i - 1) - mainIndex]
True
False
i++
string[j] == ','
False
True
string[j] == '['
True
virgulCountint++
koseliCount++
koseliIntler[i] == 0
True
False
koseliIntler[i] = koseliCount
koseliIntler[i]++
arraySize++
False
arraySize != 0
True
False
arrayCount = 1
koseliCount = 0
j++

4. ÇIKTILAR







Çıktılarda da görüldüğü gibi text dosyasından okunan kodun program çıktısı oluşmaktadır. Okunan koda göre de yer ve zaman karmaşıklıkları programda görülmektedir.

5. SONUÇ

C Dilinde dosyalama fonksiyonları ve veri yapıları bütünü ile anlaşılmış olup buradan elde edilecek yer ve zaman karmaşıklıkları için programcı hazır hale getirilmiştir. Herhangi bir algoritmanın nasıl daha rahat çalışabileceği ve daha az yer işgal ederek sürdürülebilirliği sağlanabilecektir. Projenin sonunda elde edilen kazanımlar bu yöndedir.

5. KAYNAKÇA

- https://stackoverflow.com/
- 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.programiz.com/c-programming/c-file-input-output