PROGRAMLAMA LABARATUVARI 1
2. PROJE

MUHAMMED FATİH ÖZEN – ÖMER FARUK ULUSOY
200202050 – 200202032

BİLGİSAYAR MÜHENDİSLİĞİ (İÖ)

1 ÖZET

Bu döküman programlama labaratuvarı 1 dersi 2. projesi için çözümümüzü açıklamaya yönelik oluşturulmuştur.Dökümanda projenin tanımı,arastırma yöntem ve deneysel sonuçlar,algoritmalar gibi programın oluşumunu açıklayan başlıklara yer verilmiştir.Dökümanın sonunda projeyi hazırlarken kullanılan kaynaklar bulunmaktadır.

2 Projenin Tanımı

Projede c programlama dilini kullanarak girilen bir stringin ön ek(prefix) ve son ek(suffix) eklerınden dallanmalar olusturarak bır agac verı modelı oluşturulması istenmiştir.
örnek olarak banana kelimesinin son ek ve ön eklerini inceleyelim:

banana kelimesinin son eki:

1. banana (birinci karakterden başlayan sonek)
2. anana (ikinci karakterden başlayan sonek)
3. nana (üçüncü karakterden başlayan sonek)
4. ana (dördüncü karakterden başlayan sonek)
5. na (beşinci karakterden başlayan sonek)
6. a (altıncı karakterden başlayan sonek)

banana kelimesinin ön eki:

1. b (birinci karakterde sonlanan önek)
2. ba (ikinci karakterde sonlanan önek)
3. ban (üçüncü karakterde sonlanan önek)
4. bana (dörüncü karakterde sonlanan önek)
5. banan(beşinci karakterde sonlanan önek)
6. banana(altıncı karakterde sonlanan önek)

şeklindedir.

Bir katarın (p) başka bir katar (s) içinde bulunması aslında p'nin s'in herhangi bir sonekinin öneki olmasını gerektirir. Örnek olarak ana katarı banana katarının içinde bulunup bulunmadığı bulmak için banana katarının tüm sonekleri oluşturulur ve ana katarının bu soneklerin herhangi birinin öneki olup olmadığına bakılır. Yukarıda

listelenen soneklere bakıldığında ana katarı 4. sonekin önekidir ve ana katarı banana katarının içinde yer alır. Aynı arama ane katarı için yapıldığında ise, ane katarı herhangi bir sonekin öneki olmadığı için banana katarı içinde yer almaz. Bu yaklaşımla bir katar içinde (uzunluğu n karakter olsun) başka bir kararı (uzunluğu m olsun) bulmak karmaşıklığı O(n+m).Katarlar ne kadar uzun olursa olsun sınırlı sayıda farklı karakterin birleşmesiyle oluşur. Örnek olarak çok fonksiyonlu bazı proteinlerin uzunlukları binlerce karakter olabilirken bir protein dizilimleri 20 farklı karakterden oluşur. Böyle çok uzun katarlar içinde çok kısa birçok katarı aramak yukarıda anlatılan temel yöntemi kullanılarak yapılması pahalıdır. Ancak soneklere dayalı bazı veri yapıları kullanılarak arama işlemi çok daha ucuza (algoritmik karmaşıklık olarak) yapılabilir. Bu amaçla sonek ağaçları ve sonek dizileri geliştirilmiştir.

n uzunluklu s katarın sonek ağacı aşağıdaki özelliklere sahiptir:

• Ağacın 1'den n'e kadar numaralandırılmış n adet yaprağı vardır.
• Kök dışında her düğümün en az iki çocuğu vardır.
• Her kenar s'in boş olmayan bir altkatarı ile etiketlenir.
• Aynı düğümden çıkan kenarların etiketleri farklı karakter ile başlamalıdır.
• Kökten başlayıp k. yaprağa giden yoldaki kenarların etiklerinin birleştirilmesi ile k. sonek elde edilir.

Bu projeyi gerçekleştirmek için kullanılacak fonksiyonlar:
• İLERLE();
• DALLA();
• BASTIR();
• HASSUFFİX();
• KATARARAMA();
• TEKRARIENUZUNKATAR();
• ENÇOKTEKRAREDENKATAR();
Olmak üzere 7 tanedir.

3 Araştırma Yöntem ve Deneysel Sonuçlar

bu proje için öncelikle veri yapıları ve özelliklerini araştırdık.Daha sonra istenen ağaç veri modeline baktık.Bu proje için izlediğimiz yöntem böl,parçala ve yönettir.Tüm isterleri fonksiyonlara parçalayarak mantığını kavrayıp içeriğini tek tek yazdık.Yapamadığımız fonksiyonları başka bir main dosyasında açıp orada yapıp asıl dosyaya gömdük.Projeyi yaparken kalem kağıt kullanmaya özen gösterdik.

Sonuç olarak projeden algoritma analizimizin geliştiğini fonksiyonlarla programlamaya adım atmayı gerçekleştirmeyi takım arkadaşı olarak proje yapmayı verimli bulduk.

4 Algoritma-Kod Bilgisi

Bu kısımda proje için geliştirdiğimiz algoritmayı açıklayacağız.

Bu proje için izlediğimiz yöntem daha öncede söylediğimiz gibi böl,parçala ve yö-

net prensibidir.Algoritmamızı fonksiyonlar üzerinden açıklayacağız.
Fonksiyonlara girmeden önce,fonksiyonlara ulaşmak için swicth case yapısı ile kontrol işlemi vardır ve struct yapısıyla oluşturulmuş node adında struct yapısı vardır.Bu yapının içinde kelime ve bu kelimeye ait dallanma için gerekli olan çocuk arrayi ve bağlı listeyi oluşturmak için struct türünde next pointerı vardır.

4.1 Fonksiyonlar

4.1.1 hasSuffix()

Bu fonksiyonun amacı : girilen kelimenin ön ek ve son eklerinin herhangi birinin birbirine eşit olup olmadığını kontrol ederek kelimenin son ek ağacı(suffix tree) oluşturup oluşturmadığı kontrolü yapılmıştır.

hasSuffix() fonksiyonun amacını yukarıda belirttik. Bu amaca göre fonksiyon girilen kelimeyi parametre olarak alır daha sonra kelimenin ön ek(preffix) ve son ek(suffix)lerini tutacağı string türünde preffix arrayi ve suffix arrayi oluşturduk.
Kelimenin preffix ve suffix arraylerini oluştururken strcpy() fonksiyonundan yararlandık.strcpy() fonksiyonu adressel olarak kopyalama işlemi yaptığından ve her harf bir bit olduğundan bir döngü içinde hem suffix hem preffixleri tek seferde oluşturduk.
Döngü 0'dan başlayıp kelimenin boyutu kadar ilerlemektedir her iterasyon sonucu suffix ve preffixler kopyalama işlemleri sonucu oluşturulmuştur. Suffix ve preffixler oluşturulduktan sonra birbirine eşit olup olmadığı kontrolü yapılmıştır.Bu fonksiyonun geri dönüş değeri integer olup eğer ağaç oluşturuluyorsa 0 oluşturulmuyorsa 1 olarak döndürülmüştür.

4.1.2 ilerle()

Bu fonksiyonun amacı : hasSuffix fonksiyonundan gelen geri dönüş değerine göre baş harfleri farkli olan substringleri baglı liste olarak bir yapı oluşturmaktır.
İlerle() fonksiyonunun amacını yukarıda belirttik.Bu amaca göre ilk olarak girilen kelimenin suffixlerini bir substring arrayinde tutarken kaçıncı substring olduğunuda strcat() fonksiyonuyla substring sonuna ekledik.Substringler oluşturulduktan sonra bir döngü içerisinde baş harfleri farklı olan substringleri ilerle fonksiyonuna parametre geçirdik.İlerle() fonksiyonunun içerisinde klasik bağlı liste algoritmasını uyguladık.

4.1.3 dalla()

Bu fonksiyonun amacı : hasSuffix fonksiyonundan gelen geri dönüş değerine göre baş harfleri aynı olan substringleri oluşturulmuş baglı listedeki rootların çocuklarını oluşturmaktır.
dalla() fonksiyonunun amacını yukarıda belirttik.Bu amaca göre ilerle() fonksiyonuna gönderilmeden önce oluşturulmuş substring arrayindeki baş harfleri aynı olan substringleri bu fonksiyona parametre olarak geçtik.Bu fonksiyon içerisine gelen substrinleri çocuk arrayi içerisine sıralı bir şekilde ekledik.

4.1.4 bastır()

Bu fonksiyonun amacı: oluşan ağaç yapısını görüntülemektir. bastır() fonksiyonunun amacını yukarıda belirttik. Bu amaca göre bir while() döngüsü içerisinde bağlı listedeki harfleri farklı olan root kelimeler basılmıştır.Kelimeler basılırken hemen akabinde bu kelimelerin çocuklarıda bir while döngüsü içerisinde basılmıştır ve bunları düzenli bir şekilde birleştirerek ve ascıı tablosundaki grafik şekillerinden de yararlanılarak ekrana ağaç yapısı bastırılmıştır. ilerle() ve dalla() fonksiyonlarından oluşan görüntü:

4.1.5 katarArama()

Bu fonksiyonun amacı: girilen kelimenin içerisinde kullanıcının istediği bir kataı aramaktır.
katarArama() fonksiyonunun amacını yukarıda belirttik.Bu amaca göre ilk girilen stringin boyutu kadar dönecek bir dış döngü içerisinde aranacak katarın boyutu kadar dönecek iç döngü oluşturulmuştur.Bu iç döngü içerisinde harfler teker teker kontrol edilerek katar içerisinde var olup olmadığı anlaşılmıştır. bu şekilde yazdırılmıştır:

4.1.6 tekrarıEnUzunKatar()

Bu fonksiyonun amacı: girilen string içerisindeki tekrar eden en uzun katarı bulmaktır.
tekrarıEnUzunKatar() fonksiyonunun amacını yukarıda belirttik.Bu amaca göre bir tane enuzun isminde içinde boşluk olan bir string tanımladık.İlk stringin boyutu kadar dönecek bir dış döngü içerisinde dış döngünün bir fazlasından başlayarak yine boyuta kadar dönecek iç döngü oluşturulmuştur.Bu iç döngü içerisinde iki tane kontrol amaçlı test1 ve test2 isimli stringler tanımlanmıştır test1 stringi her dış döngü iterasyonu sonucu oluşan son eki tutmakta olup test2 stringi ise iç döngüde oluşan son ekleri tutmaktadır.Bu iç döngüler içerisinde test1 ve test2 nin her harfini tek tek kontrol edecek bir döngü oluşturulmuştur.Bu döngü içerisinde test1 ve test2 nin harflerinin farklı olduğu harfe kadarki stringi bir tmp değişkenine kopyaladık daha sonra tmp değişkeninin boyutu ile enuzun isimli stringin boyutlarını karşılaştırdık büyük olanı enuzun'a kopyaladık.

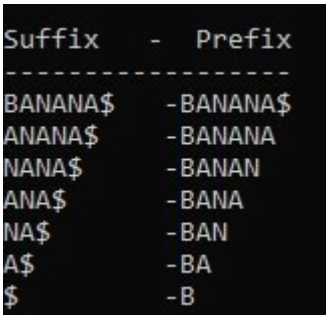
4.1.7 enÇokTekrarEdenKatar()

Bu fonksiyonun amacı: girilen string içerisindeki en çok tekrar eden katarı bulmaktır.
enÇokTekrarEdenKatar() fonksiyonunun amacını yukarıda belirttik.Bu amaca göre bir tane ençok isminde içinde boşluk olan bir string tanımladık.İlk stringin boyutu kadar dönecek bir dış döngü içerisinde oluşturulan her suffixin 1 den başlayarak bu suffixe kadar dönen bir iç döngü oluşturulmuştur.Bu döngü içerisinde dış döngüden gelen suffixlerin her iterasyon sonucu 1'den başlayıp boyutu artırılarak bir tmp değişkenine atanmıştır.Bu döngü içerisinde kontroller yapılarak en ok tekrar eden katar bulunmuştur.

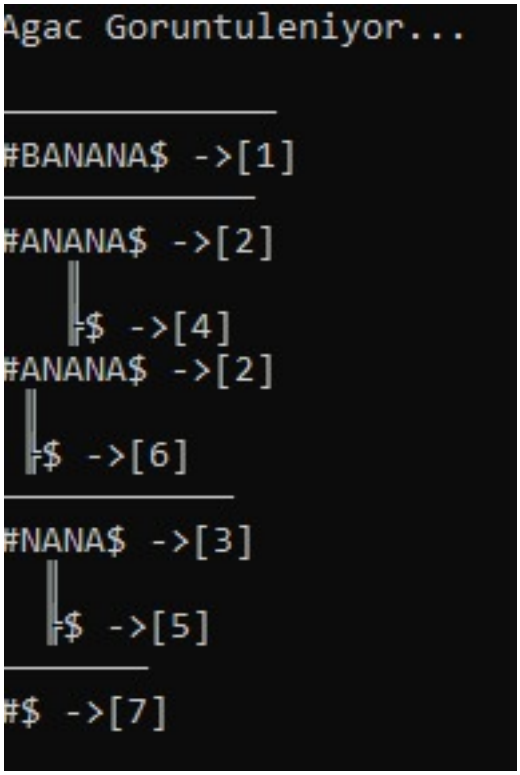
5 Yalancı Kod

Başla
(öncelikle bir txt dosyasının içine katar girilir)
(bir sonsuz döngü içerisinde çıkış yapana kadar sürekli seçenekleri sunar)
yaz(secimler)
yaz(1->Suffix Tree olusturulabilirmi.)
yaz(2->girilen katar içerisinde katar arama)
yaz(3->girilen katar içerisinde tekrar eden en uzun katarı bulma)
yaz(4->girilen katar içerisinde en çok tekrar eden katarı bulma)
yaz(5->çıkış)
yaz(bir seçim yapınız)
oku(seçim)
secim 1 ise;
hasSuffix fonksiyonuna gider suffix ve prefixlerini karşılaştırır eğer aynı suffix ve preffix var ise
yaz(ağaç oluşturulamaz)
eğer hiç eşit suffix ve preffix yok ise
yaz(hiçbir suffix ve preffix eşit olmadığından ağaç oluşturulabilir)
fonksiyondan çıkar
dosya içine girilen kelimenin boyutu kadar dönecek bir döngü içerisinde:
harfleri tek tek karşılaştırarak dallanmalar oluşturulur
her dalın kaçıncı substring olduğunu yanına yazdırır
ardından bastır fonksiyonuna gider
yaz(ağaç görüntüleniyor...)
ascıı tablosunu kullanarak düzenli bir grafik şeklinde ağaç oluşturur
seçim 2 ise:
katar arama fonksiyonuna gider ve asıl katar içinde aranması istenen katarın harflerini karşılaştırır
aranana katar var ise:
yaz(aranan katar bulundu asıl katar içinde n defa geçiyor)
aranan katar yok ise:
yaz(aranan katar asıl katar içinde bulunamamıştır.)
seçim 3 ise:
tekrariEnUzunKatar fonksiyonuna gider ve katar içinde tekrar eden katarları bulur ardından bu katarları kendi aralarında uzunluk kıyaslamasına tabi tutar ve son olarak en uzun katarı yazar)
seçim 4 ise:
enÇokTekrarEdenKatar fonksiyonuna gider ve tekrar eden katarları bulur ardından bu katarları kendi arasında tekrar etme sırasına göre sıralar ve en çok tekrar edeni ekrana bastırır)
seçim 5 ise:
yaz(programdan çıkılıyor)
ve programdan çıkar
seçim default :
yani farklı bir seçim yapılmışsa
yaz(hatalı seçim)
tekrar tekrar seçim ekranına döner taaki doğru seçimlerden birini yapana kadar
bitir

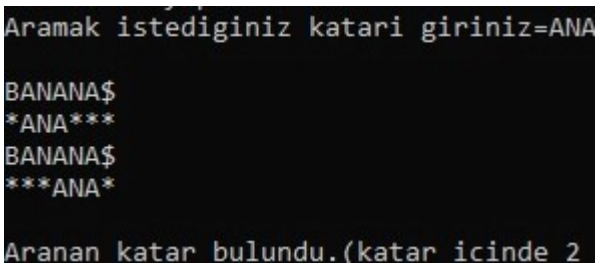
6 Ekran Çıktıları



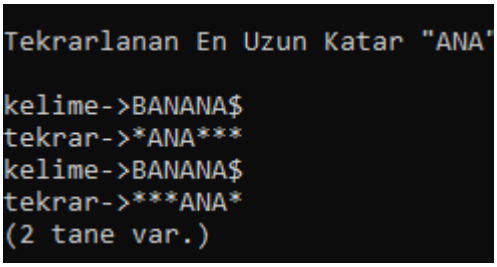
Şekil 1: HasSuffix görüntüsü



Şekil 2: ağaç görüntüsü



Şekil 3: arama görüntüsü



Şekil 4: en uzun katar

7 Kaynakça

http://matematik.fef.duzce.edu.tr/Dokumanlar/matematik_fef/latex_notlarrr.pdf
https://www.latexceviri.com/latex-dokumana-resim-grafik-veya-sekil-ekleme