BLM209 PROGRAMLAMA LABORATUVARI I PROJE 2

Barış Kakilli
Kocaeli Üniversitesi
Bilgisayar Mühendisliği Bölümü
200201012@kocaeli.edu.tr

Muhammed Sina Çimen
Kocaeli Üniversitesi
Bilgisayar Mühendisliği Bölümü
200201032@kocaeli.edu.tr

I. Özet

Bu projenin amacı sonek ağaçlarını ve sonek dizililerini kullanarak karakter dizileri üzerinde s karakter dizisi için sonek ağacı oluşturulabilir mi, sonek ağacı oluşturulan bir s karakter dizisi içinde p karakter dizisi geçiyor mu, geçiyorsa ilk bulunduğu yerin başlangıç pozisyonu nedir, kaç kez tekrar etmektedir, sonek ağacı oluşturulan bir s karakter dizisi içinde tekrar eden en uzun altkarakter dizisi nedir, kaç kez tekrar etmektedir ve sonek ağacı oluşturulan bir s karakter dizisi içinde en çok tekrar eden altkarakter dizisi nedir, kaç kez tekrar etmektedir gibi işlemleri kontrol ederek grafiksel olarak bunu göstermektir. Bu isterlerin C programlama dili kullanılarak yerine getirilmesi amaçlanmıştır.

II. Giriş

Biyoinformatik'de adı sıkça geçen algoritmalardan olan sonek ağac(suffix tree) veri yapısı bir dizgi model(pattern) eşleştirme algoritmasıdır. Örneğin elinizde uzun bir karakter dizisi olsun ve siz bu karakter dizisi içinde alt karakter dizilerini aramak ve hatta bu karakter dizilerinden kaç adet bulunduğunu öğrenmek istiyorsunuz. İşte bu veri yapısı bu işlemleri kolaylaştırmak ile birlikte gayet hızlı işlem yapmamıza olanak sağlıyor.
Bu tip bir yöntem kullanmadan ilkel olarak çözüm olarak şunu yapabilirdik. Ana dizgi içerisinde lineer olarak sırayla karşılaştırma yapa yapa devam ederek sonuca ulaşabilirdik. Bu bir yöntemdir fakat karakter dizisi boyutları arttıkça bu işlem çok maliyetli olmaktadır.

Bir karakter dizisi için sonek, karakter dizisi herhangi bir karakterinden başlayarak sonuna kadar olan kısımdır. Örnek olarak "banana" kelimesinin tüm sonekleri aşağıdaki gibidir.

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)

Bir kelimenin öneki ise kelimenin ilk karakterinden başlayarak herhangi bir karakterine kadar olan kısımdır. Örnek olarak banana kelimesinin tüm önekleri aşağıdaki gibidir.

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

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

listelenen soneklere bakıldığında ana karakter dizisi 2. sonekin önekidir ve ana karakter dizisi banana karakter dizisinın içinde yer alır. Aynı arama ani karakter dizisi için yapıldığında ise, ani karakter dizisi herhangi bir sonekin öneki olmadığı için banana karakter dizisi içinde yer almaz. Bu yaklaşımla bir karakter dizisi içinde (uzunluğu n karakter olsun) başka bir kararı (uzunluğu m olsun) bulmanın karmaşıklığı O(n+m)'dir.
Karakter dizileri 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 karakter dizisi 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 karakter dizisin 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 altkarakter dizisi 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.

Örnek olarak "banana$" karakter dizisinin sonek ağacı Fig. 1'de verilmiştir.

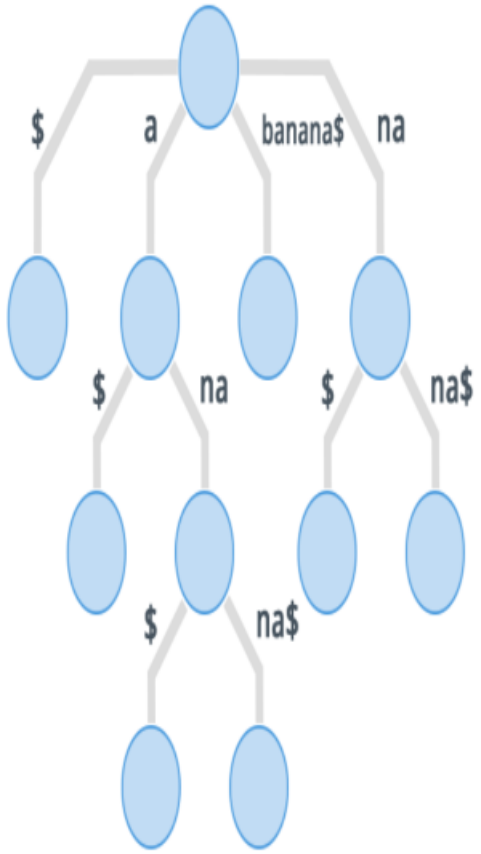

Fig. 1. Örnek Sonek Ağacı

Yukarıdaki görselde "banana" karakter dizisini "banana$" şeklinde oluşturmamızın sebebi ise sonek ağacı oluşumunu garantiye almaktır. Bu ne demek şöyle açıklayalım. Her karakter dizisi bir sonek ağacı sahibi değildir. Eğer bir karakter dizisinin sonekleri öneklerine eşitse bir sonek ağacı yoktur. Mesela "cdbcd" için inceleme yapalım.

Sonekleri : cdbcd,dbcd,bcd,cd,d
ve
Önekleri : c,cd,cdb,cdbc,cdbcd şeklindedir.

Dizginin kendisi dikkate alınmadan incelendiğinde "cd" değerlerinin ortak olduğu görülmektedir. Dolayısı ile "cdbcd" için bir sonek ağacı çizilemez. Bu sorunu sadece "cdbcd" karakter dizisinin sonuna bir "$" ekleyerek çözebiliriz. Eğer "cdbcd$" için inceleme yaparsanız bir sorun olmadığını göreceksiniz.

III. YÖNTEM

Yöntem olarak ilk başta karakter dizimizi bir satırlık bir text dosyasından okuduk ve bu karakter dizimizin boyutunu hesaplamayı tercih ettik. Karakter dizisinin boyutunu kullanarak bir dizi yapısı oluşturduk(dizi[3][harfsayisi]). Böyle dizi yapısı kullanmamızın sebebi ise projenin ileri kısmındaki isterleri yerine getirebilmek için gerekli verileri dizilerin elemanlarında kullanabilmek. Önbellek kaynaklı oluşabilecek hataları engellemek için de her bir dizi değerimize -1 değerini verdik. dizinin ilk kısmını harflerin indexlerini saklayacak şekilde tasarladık. Dizinin ikinci verileri ise o harf tekrar ediyorsa o harfin ilk geçtiği yeri gösterecek değilse -1 olarak kalacak. Dizinin üçüncü kısmı tekrar eden harflerin kaç adet olduğunu gösterecek eğer tekrar etmiyorsa -1, ana kısım ise 0 olarak değiştirilecek. Örnek: abcabx için 2. a nın dizi[0] dizi[1] ve dizi [2] değerleri 3,0,2 olacak.

```
0 1 2 3 4 5 6 7 8 9
0 1 2 0 1 -1 0 1 2 -1
0 0 0 2 1 -1 3 2 1 -1
```

Fig. 2. abcabxabcd için dizinin çalışmasına örnek

İsterlere geldiğimizde ise 1.isterin çalışma mantığı fazlasıyla basit; karakter dizisini döngülerle bi tersen bir de düzden okuyoruz ve bu değerleri kıyaslıyoruz eşit değerler bulunursa sonek ağacı oluşturulamayacağını döndürüyoruz. Bulunamazsa zaten oluşturulabilirdir.

2.isteri gerçekleştirmek adına strstr() isimli bir fonksiyondan yararlandık. Bu fonksiyon aynı zamanda bize ilgili karakter dizisinin nerede bulunduğunu da döndürüyor. Kaç kez tekrar ettiğini de sayarak ekrana yazdırdık.

3. istere gelirsek burada kullandığımız algoritma ise şöyle çalışmakta : en uzun tekrar eden karakter dizisini bulmak için ilk önce dizinin üçüncü kısmına bakıyoruz ve en büyük değeri bir değişkene atıyoruz bu bizim en uzun karakter dizimiz için [2][i] değerini tutuyor buradan normal i yi buluyoruz bir daha döngüye girip ve i değerinden itibaren karakter dizisini yazdırıyoruz.

4. isterde ise dizi[1]de en çok tekrar eden adımı bulduktan sonra bu bulduğumuz sonuç tek bir harf olduğundan bulunan her sonuç için sonraki adımları birbiriyle aynı mı diye kontrol ediyoruz eğer aynı ise sonucumuz o oluyor. Örnek: dabxdabczabc için ab, abc ve abc mevcut bu üçünden en kısa olanı ab olduğu için cevabımız ab oluyor.

IV. DENEYSEL SONUÇLAR

```
Islem yapmak istediginiz dosya ismini giriniz: katar.txt
katar.txt icerigi: abcabxabcd

0 | Cikis Yap
1 | s katari icin sonek agaci olusturulabilir mi kontrolu
2 | s katarindaki p katarini bulma
3 | s katari icinde tekrar eden en uzun altkatar
4 | s katari icinde en cok tekrar eden altkatar
```

Fig. 3. Menü

```
Hangi Islemi Yapmak Istediginizi Seciniz: 1
abca icin sonek agaci olusturulamaz.
```

Fig. 4. 1.ister için 1.örnek

```
Hangi Islemi Yapmak Istediginizi Seciniz: 1
abcabxabcd icin sonek agaci olusturulabilir.
```

Fig. 5. 1.ister için 2.örnek

```
Hangi Islemi Yapmak Istediginizi Seciniz: 2
bulmak istediginiz katari giriniz: ab
evet katar geciyor.
katarin baslangic indexi: 0
3 kez geciyor.
```

Fig. 6. 2.ister için örnek

-

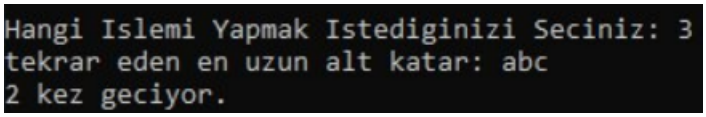

Fig. 7. 3.ister için örnek

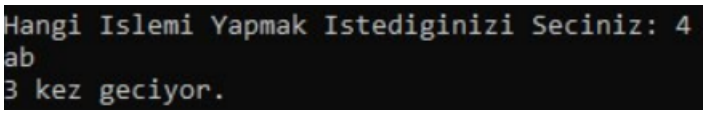

Fig. 8. 4.ister için örnek

V. Sonuç

Projede bizden istenen tüm isterler eksiksiz bir şekilde başarıyla tamamlanmıştır.

VI. Yalanci Kod

1.Başla
2.3'e harf sayisi boyutlarında bir dizi al
3.bu dizinin elemanlarını yukarıda açıklanan şekilde doldur
4.bu diziden ikinci kısmı -1 olan sonekleri hemen yaz(bu sonekler diğerleriyle bağlantısı olmadığından köke bağlı) burdan sonra döngüye girip ikinci kısmı 0dan büyük olmayan sayı kalana kadar devam et
5.benzerlik olanlarda ikinci kısımları eşit olanları al burdan üçüncü kısmı en az olanı(yani en az karakter bağlantısı olanı) ekrana o kadar karakteri yaz
6.alt kısma geçip ağacı böl
7.üçüncü kısmı daha büyük olan sayının kalan kısmını yazıyoruz
8.diğer kısıma kısa olan 5. kısımda yazdığımız soneki tamamlıyoruz.
9.7. kısımda kalan kısmını yazdıktan sonra bir alt satıra geçip böl
10.bir kısmına ana soneki yazarken(üçüncü kısmı 0 olan). 11. diğer kısmına 7. kısımda yazıp tamamlamadığımız soneki tamamlıyoruz
12. böylelikle ikinci kısımda o sayıya ait tüm sonekler tamamlanmış oluyor
13. ikinci kısmı döngüde tekrar okurken aynı ağacı bir daha oluşturmamak için yaptığımız tüm değerleri -2ye çeviriyoruz.

VII. Kaynakça

[1] https://stackoverflow.com/
[2] https://www.geeksforgeeks.org/
[3] https://www.koseburak.net/blog/suffix-tree/
[4] https://www.hackerearth.com/practice/data-structures/advanced-data-structures/suffix-trees/tutorial/
[5] https://en.wikipedia.org/wiki/Suffix_array
[6] https://en.wikipedia.org/wiki/Suffix_tree
[7] https://www.hackerearth.com/practice/data-structures/advanced-data-structures/suffix-arrays/tutorial/
[8] https://youtu.be/UbhlOk7vjVY
[9] https://web.stanford.edu/ mjkay/suffix_tree.pdf