PROGRAMLAMA LABORTUVARI PROJE 2

Alperen KOLAT
Kocaeli Üniversitesi Mühendislik Fakültesi,
Bilgisayar Mühendisliği Bölümü
200201006@kocaeli.edu.tr

Beyhan KARADAĞ
Kocaeli Üniversitesi Mühendislik Fakültesi,
Bilgisayar Mühendisliği Bölümü
190201012@kocaeli.edu.tr

***Abstract*—Sonek ağaçları genellikle biyoinformatik uygulamalarda kullanılır, DNA veya protein dizilerindeki (uzun karakter dizileri olarak görülebilen) kalıpları arar . Uyumsuzluklarla verimli bir şekilde arama yapma yeteneği, en büyük güçleri olarak kabul edilebilir. Sonek ağaçları da veri sıkıştırmada kullanılır ; tekrarlanan verileri bulmak için kullanılabilirler ve Burrows-Wheeler dönüşümünün sıralama aşaması için kullanılabilirler . LZW sıkıştırma şemalarının varyantları, sonek ağaçlarını (LZSS) kullanır. Bazı arama motorlarında kullanılan bir veri kümeleme algoritması olan sonek ağacı kümelemesinde bir sonek ağacı da kullanılır [1]. Bu proje ile bir sonek ağaçı oluşturduk ve oluşturduğumuz yapı üzerinde arama işlemleri yaptık aynı zamanda bu işlemleri de dinamik olarak grafiğe dökülmesi işlemlerini gerçekleştirdik.**

***Index Terms*—**

I. GIRIŞ

Çalışmamızın grafik kısmında Allegro kütüphanesinden yararlandık. Kodumuzu önceleri modüler yapmak ve derleme işlemlerini kolaylaştırmak amacıyla makefile dosyası oluşturduk anca bu dosyadan sadece derleme komutunu kolaylaştırma amacıyla kullandık. Proje, grafiklerin çizildiği kısım ve ağacın bellek üzerinde oluşturup üzerinde işlemlerin yapıldığı kısım olmak üzere 2 temel kısımdan oluşmaktadır. Öncelikle bellek üzerinde ağaç yapısını oluşturduk daha sonra oluşan düğümler üzerinde özyinelemeli şekilde dolaşan bir fonksiyon ile ağaç üzeridne dolaşarak ekrana yansıtılacak düğümlerin ve onları birbirine bağlayan dalların kordinatları belirlenmektedir. Ağaç üzerinde dolaşım tamamlandıktan sonra artık ekrana çizim kısmı başlamaktadır bunu saniyede 30 kere ile yapmaktayız. "Q" tuşu ile arayüzümüze bir kesme sinyali verilmektedir bu tuş tetiklendiğiinde komut satırı üzerinden ulaşılabilen menümüz ile başlangıçta oluşturduğumuz ağaç üzerinde arama işlemleri yapıyor ve grafik arayüzü ile etkileşime girerek hangilerini boyayacağımızı kontrol ediyoruz. Bu şekilde arama işlemini ve bunu görselleştirme işlemlerini yapıyoruz.

II. YÖNTEM

Kullanıcı programı çalıştırdıktan sonra text dosyasından katar okunur.Katar `tree_control` fonksiyonuna gönderilir. katarı pointer aritmetiğiyle başından ve sonundan kataın içinde gezerken iki farklı katara atanır bu iki katarda eşleme varsa ağaç oluşturulamaz. Bu kısımdan sonra ağacın bellek üzerindeki yerleşim işlemi başlatılır. Pointer aritmetiği kullanarak her seferinde başlangıç stringinden 1 karakter atılarak find_branch fonksiyonuna gönderilir. Bu fonksiyon kök üzerindeki dallarda var olan string ile bir eşleşme arar şayet eşleşme bulamazsa buraya yeni bir dal ekler eşleşme bulması durumunda ise kaç karakterlik bir eşleşme olduğunu tespit eder ve regulation_tree() fonksiyonunu çağırır. Bu fonksiyon artık kökte değil yeni bir dal üzerindedir öncelikle bu yeni dalda bir işleşme olup olmadığını arar. Eşleşme olamsı durumunda eşleşme bulunan dala kendisini çağırır ve kapanır. Şayet eşleşme olmazsa yeni stringin buraya ekleneceği anlaşılır. Bu ekleme işleminin yapılması için bellekten 2 yeni yer tahsil edilir. Bu alınan yerler şu anki dalın yeni bağlantılarını oluşturur ve hali hazırda var olan dalın stringi az önceki eşlenen karakterler hariç silinir. Böylece sanki bir araya ekleme işlemi yapılmış gibi olur. Ağaç bellek üzerinde oluşturulduktan sonra allegro ile çizim hazırlıkları yapılır. Kütüphanenin gerekli ayarları yapıldıktan sonra öncelikle kökü çizmek için show_root() fonksiyonu çağrılır. Bu fonksiyon ona verilen başlangıç noktası ile kökü ve ona bağlı olan dalları çizerek içlerine stringleri yerleştirir. Daha sonradan trigonometri yardımıyla bir sonraki dalların çizim işlemine devam edeceği kordinatları hesaplayarak onları çağırır ve her fonksiyon kendi bulunduğu düğümdeki çizimleri yapana kadar kendini çağırır.

Ekrana sunulan grafik çiziminde kullanıcı "w,a,s,d" tuşları ile zoom yapar yön tuşlarıyla yardımıyla da oluşturulan ağaç üzerinde gezinebilir. Daha sonra "q" tuşuna basarak grafik çizme durdurulur ve komut satırı çalışır. Kullanıcı menüden fonksiyon seçer ve aşağıda bulunan durumlar gerçekleşir menüden seçilen fonksiyonlar grafik kısmına düğüm adresleri göndererek çizilen ağaç üzerinde onların renkli görünmesini sağlarlar.

A. Aranan Katarı Bulma

Kullanıcı aranan katarı komut satırına girer. aranan katar `substring_check()` fonksiyonuna gönderilir. Fonksiyon ağacı gezerek aranan katarı bulmaya çalışır. Katar sonek ağacında bulunmuyorsa `"bulanamadı"` diye ekrana bastırılır.

Katar sonek ağacında varsa bulunduğu dalın düğümlerinin adreslerini `adres` dizisinde tutulur. `adres` dizisinde tutulan adreslerin karşılık geldiği düğümler grafik üzerinde boyanır. Aynı anda `addresss` dizisinde son

düğümün adresi tutulur . `addresss` dizisinde tutulan adres `node_counting()` fonksiyonuna gönderilir. `node_counting()` fonksiyonuna gönderilen adresin yaprağı var mı kontrol edelir yaprak sayılır ve aranan kataın kaç kez tekrar etttiği bulunur ve ekrana basılır.

B. En Uzun Tekrar Eden Katarı Bulma

`longest` adlı boş katar tanımlanır. katar `longest_find` adlı fonksiyona gönderilir. fonksiyon tekrar eden dalın katarı ile `longest` uzunluğu karşılaştırır. eğer `longest`'dan uzunsa `longest`'a atanır. `addresss` dizisinde son düğümün adresi tutulur `longest`'katarı varsa `substring_check()` fonksiyonuna gönderilir. Fonksiyon ağacı gezerek `longest` katarını bulmaya çalışır. Bulunca bulunduğu dalın düğümlerinin adreslerini `adres` dizisinde tutar. Tutulan adreslerin karşılık geldiği düğümler grafik üzerinde boyanır. `addresss` dizisinde son düğümün adresi `node_counting()` gönderilir ve ordaki yapraklar sayılır. Sayma sonuçunda elde edilen tekrar sayısı ekranan basılır.

C. En Çok Tekrar Eden Katarı Bulma

`node_counting_caller` fonksiyonuna kökün adresi gönderilir. kökten çıkan düğümler sırasıyla `node_counting` fonksiyonuna gönderilir. `node_counting` fonksiyonu düğümün yapraklarını sayar ve fonksiyonu bitirir.

`node_counting_caller` fonksiyonu Sayılan dal sayısını `most_repaet` adlı diziye atar. dizinin en büyük dal sayısı bulunur hem adresi `adres` adlı diziye atılır hem de tekrar sayısı ve kataın kendisi ekranana basılır . `adres` dizisinde tutulan adreslerin karşılık geldiği düğümler grafik üzerinde boyanır.

III. Sonuç

Proje kapsamında suffix ağaçlarının kullanımını kavrayarak daha sonra karşılaşacağımız problemlerde de kullabileceğimiz bir yapı öğrendik. Grafik kütüphanesinin çalışma şekilini de görmüş olduk. Çözemediğimiz tek sorun grafik kütüphanesinin bazen tuşu basılı olarak algılaması oldu bu sebepten bazı çağrılarımıza yanıtsız kalabilmekte.

Kaynakça

- https://stackoverflow.com/questions/1701728/graphics-library-in-c
- http://www.belgeler.org/howto/makefile-nasil-kullanimi.html
- https://gitlab.com/limdingwen/6502js-but-c
- https://www.ncbi.nlm.nih.gov/nuccore/NC_045512
- https://github.com/liballeg/allegro_wiki/wiki/Allegro-Vivace
- https://man7.org/linux/man-pages/man1/man.1.html

References

[1] Zamir, Ören; Etzioni, Oren (1998), "Web belgesi kümeleme: bir fizibilite gösterimi", SIGIR '98: Bilgi erişiminde araştırma ve geliştirme üzerine 21. yıllık uluslararası ACM SIGIR konferansının bildirileri , New York, NY, ABD: ACM, s. 46 –54.

IV. Akış Diyagramları

A. tree_control()

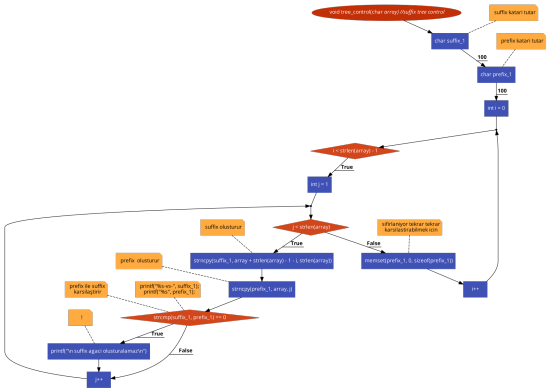


Fig. 1. tree_control fonksiyonunun akış şeması

B. longest_find()

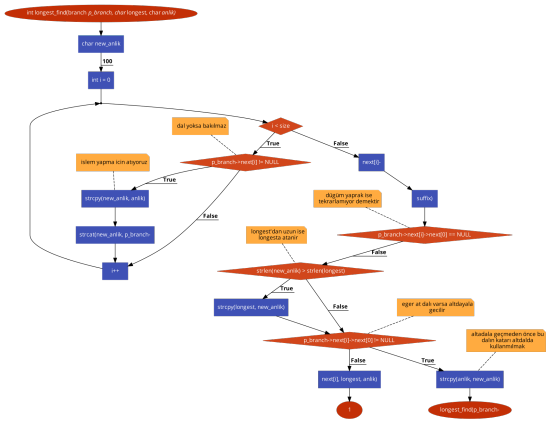


Fig. 2. Longest_find fonksiyonunun akış şeması

C. find_branch()

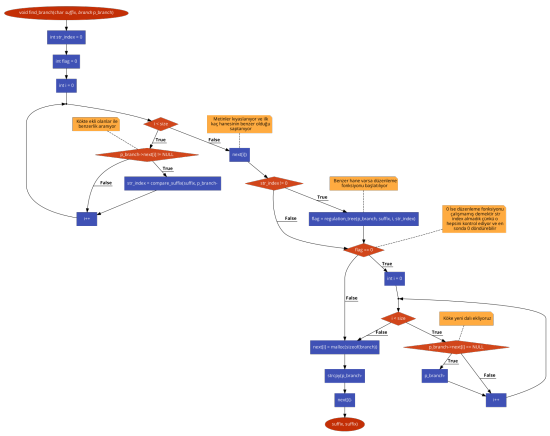


Fig. 3. find_branch fonksiyonunun akış şeması.

D. node_counting_caller()

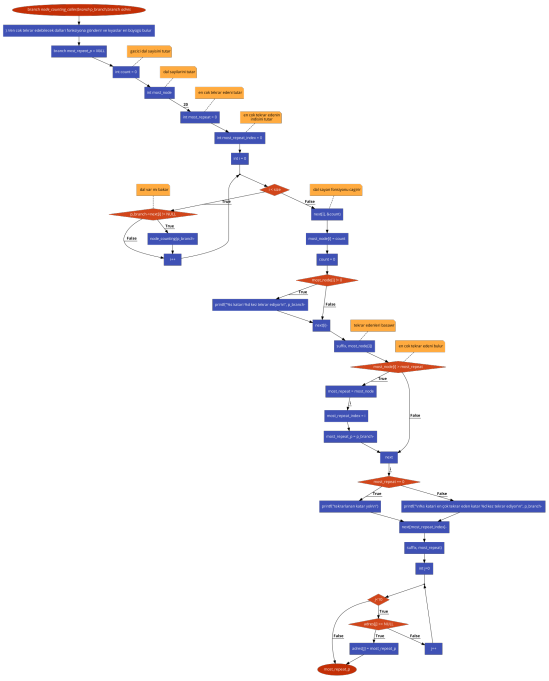
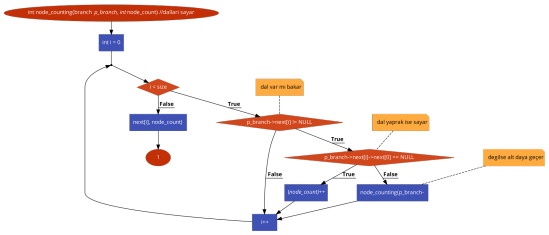
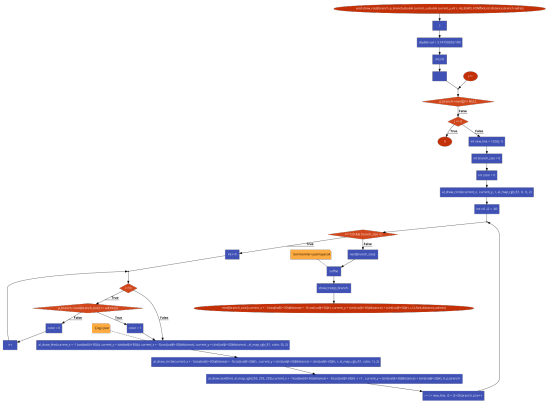


Fig. 4. 3. node_counting_caller fonksiyonu labeloverflow

E. node_counting()



F. show_root()



V. Deneysel Sonuçlarlardan Kareler

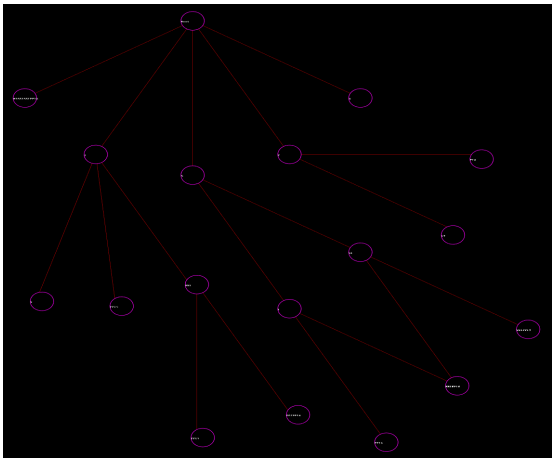


Fig. 7. İlk Çalışma Görüntüsü

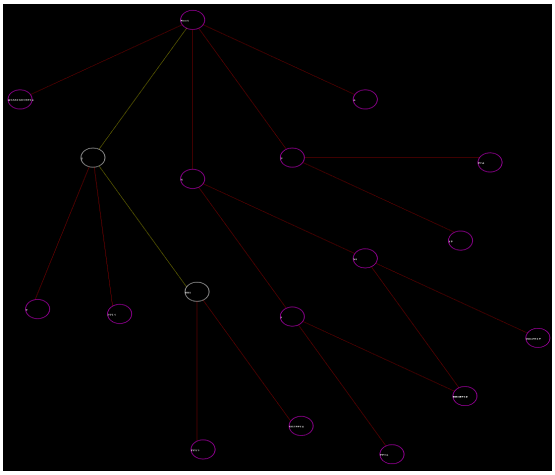


Fig. 8. Örnek Bir boyama

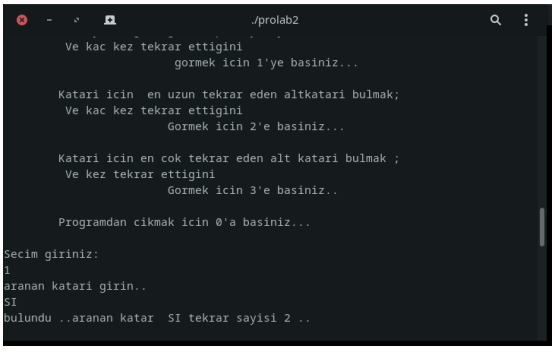


Fig. 9. Komut Menüsü

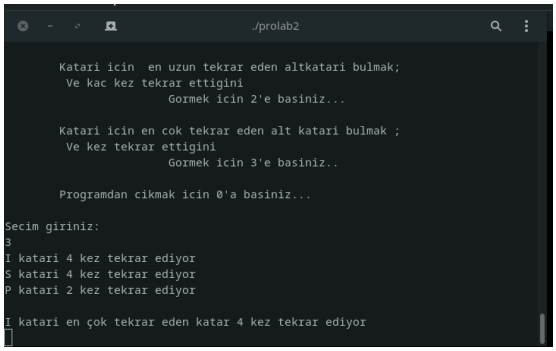


Fig. 10. 3. İster sonucu