Kocaeli Üniversitesi
Bilgisayar Mühendisliği Bölümü
Programlama Laboratuvarı 1
Proje 2

Robera Tadesse GOBOSHO
ID:190201141
robtad318@gmail.com

Muhammad Abdan SYAKURA
ID:200201147
prof.syakur@gmail.com

ÖZET

Bu projenin amacı sonek ağaçlarını ve sonek katarlilerini kullanarak katarlar üzerinde bazı arama işlemleri yapmaktır. Bir katar için sonek, kataın herhangi bir karakterinden başlayarak sonuna kadar olan kısımdır. Sonek ağaçları doğal olarak asimetriktir: önek uzantıları yalnızca birkaç güncellemeye neden olurken, sonek uzantıları tüm son ekleri etkiler ve bir güncelleme dalgasına neden olur. Sonek ağacının kenarlarını, örtük düğümlerin aynı davranışı sergilediği, bantlarin verilen benzer kenarların koleksiyonlarına ayırdık ve örtük düğümlerin bantlar içinde ”yüzmesine” izin vermek için açık uçlu kenarlar kavramını kullandık. İç örtük düğümler, açık ek ağaç düğümleri ile birbirinden ayrılır. Bu özellikler, örtük düğüm güncellemelerinin dalgalarını takip etmek ve son ek ağacını amortize edilmiş cizgi olarak oluşturmak için kullanıldi.
Bu proje kapsamında sonek ağaçları kullanılarak s katarı için sonek ağacı oluşturulabilir, Sonek ağacı oluşturulan bir s katarı içinde p katarı geçip gecmedigi kontrol edilecek, geçiyorsa ilk bulunduğu yerin başlangıç pozisyonu ve kaç kez tekrarlandığı görüntülenecektir,Sonek ağacı oluşturulan bir s katarı içinde tekrar eden en uzun altkatar bulunacak ve ekrana kaç kez tekrarlandığı yazılacak, Sonek ağacı oluşturulan bir s katarı içinde en çok tekrar eden altkatar bulunup, kaç kez tekrarlandığı ekrana yazilacak.

GİRİŞ

Bir rn-karakter katarı S için bir ağaç son eki, 1’den m’ye kadar numaralandırılmış tam olarak m yaprağı olan köklü, yönlendirilmiş bir ağaçtır. Kök dışındaki her bir dahili düğümün en az iki çocuğu vardır ve her kenar boş olmayan bir S alt katarı ile etiketlenir. Bir düğümün hiçbir iki kenarı aynı karakterle başlayan kenar etiketlerine sahip olamaz. Sonek ağacının temel özelliği, herhangi bir yaprak I için, kökten I yaprağına giden yoldaki kenar etiketlerinin sıralanması, i konumunda başlayan S son ekini tam olarak heceler. Yani S[i..m]’yi heceliyor. Örneğin, xabxac katarı için son ek ağacı Şekil:1’de gösterilmiştir. Kökten I numaralı yaprağa giden yol tam dava S = xabsac’ı, 5 numaralı yaprağa giden yol ise ac son ekini heceler. S’nin 5. konumunda başlar.Yukarıda belirtildiği gibi, S için bir sonek ağacının tanımı,

herhangi bir katar için bir sonek ağacının gerçekten var olduğunu garanti etmez.

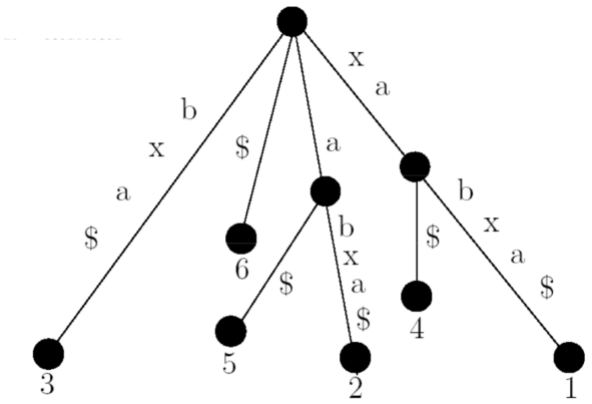

Fig. 1. xabxa katarı için sonek ağacı

YÖNTEM

A. *Tanım*

Dize için sonek ağacı S uzunluk nşöyle bir ağaç olarak tanımlanır:

- Ağacın numaralandırılmış tam olarak n yaprağı vardır 1 için n.
- Kök dışında, her dahili düğümün en az iki çocuğu vardır.
- Her kenar, boş olmayan bir alt dize ile etiketlenir. S.
- Bir düğümden başlayan hiçbir iki kenar, aynı karakterle başlayan dize etiketlerine sahip olamaz.
- Kökten yaprağa giden yolda bulunan tüm dize etiketlerini birleştirerek elde edilen dize i son eki heceler S[i..n],için i itibaren 1 için n.

Tüm dizeler için böyle bir ağaç bulunmadığından, Sdizede görülmeyen bir uçbirim simgesiyle doldurulur (genellikle $.)

Örnek olarak şekil 2'yi alırsak, "BANANA" kelimesinin sonek ağacı. Her alt dize özel karakter $ ile sonlandırılır. Kökten yapraklara giden altı yol (kutular olarak gösterilir) altı son eke karşılık gelir A$, NA$, ANA$, NANA$, ANANA$ ve BANANA$. Yapraklardaki sayılar, ilgili son ekin başlangıç konumunu verir. Kesik çizgilerle çizilmiş son ek bağlantıları inşaat sırasında kullanılır.

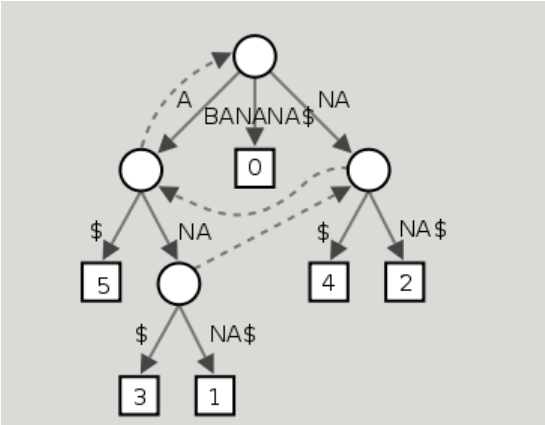

Fig. 2. "BANANA" katarı için sonek ağacı diyagramı

B. *Örtülü sonek ağacı oluşturma*

bu projede, sonuncusu S katarının gerçek bir sonek ağacına dönüştürülen bir katar örtülü sonek ağacı kullanıldı. S katarı için bir örtülü sonek ağacı, S$ için sonek ağacından, $ terminal sembolünün her kopyasını ağacın kenar etiketlerinden kaldırarak, ardından etiketi olmayan herhangi bir kenarı kaldırarak ve ardından en az iki çocuğu olmayan herhangi bir düğümün kaldırarak elde edilen bir ağaçtır. S'nin S[l..i] öneki için örtülü bir sonek ağacı, S[1..i]$ için sonek ağacı alınarak ve yukarıda açıklandığı gibi $ sembollerini, kenarları ve düğümleri silerek benzer şekilde tanımlanır. S[1..i] katarının örtülü sonek ağacını I i için 1'den m'ye kadar gösteriyoruz. Herhangi bir S katarı için örtülü sonek ağacı, yalnızca S'nin son eklerinden en az birinin başka bir sonekin öneki olması durumunda, S$ katarı için sonek ağacından

daha az yaprağa sahip olacaktır.

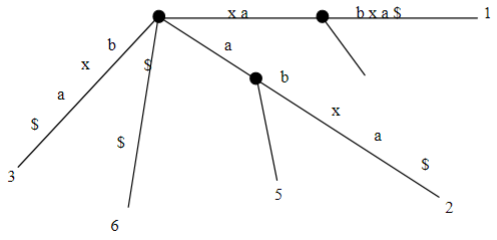

Fig. 3. xabxa$ katarı için sonek ağaç diyagramı

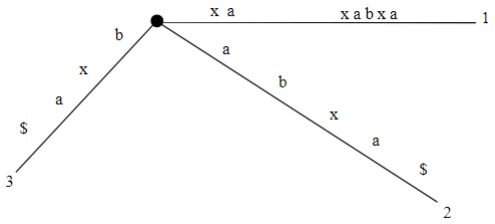

Fig. 4. xabxa$ katarı için örtülü sonek ağaç diyagramı

Bu durumdan kaçınmak için tam olarak S'nin sonuna terminal sembolü $ eklendi. Bununla birlikte, S, S'de başka hiçbir yerde görünmeyen bir karakterle biterse, o zaman S'nin örtülü sonek ağacının her sonek için bir yaprağı olacaktır ve bu nedenle gerçek bir sonek ağacı olacaktır. Örnek olarak, Şekil 3'te gösterilen xabxa$ sting için xa soneki, xabxa son ekinin bir önekidir ve benzer şekilde a dizgisi de abxa'nın bir önekidir. Bu nedenle, xabxa için sonek ağacında, 4 ve 5 numaralı yapraklara giden kenarlar yalnızca $ ile etiketlenir. Bu kenarların kaldırılması, her biri yalnızca bir çocuklu iki düğüm oluşturur ve daha sonra bunlar da kaldırılır. xabxa için elde edilen örtük son ek ağacı Şekil 4'te gösterilmektedir. Bir örtülü sonek ağacının her bir sonek için bir yaprağı olmasa da, S'nin tüm soneklerini kodlar. Her sonek, örtülü sonek ağacının kökünden gelen bir yoldaki karakterler tarafından hecelenir. Ancak yol bir yaprakta bitmiyorsa yolun sonunu gösteren bir işaret olmayacaktır. Bu nedenle, örtülü sonek ağaçları, kendi başlarına, gerçek sonek ağaçlarından biraz daha az bilgilendiricidir. örtülü sonek ağacını S için gerçek sonek ağacını elde etmek için Ukkonen'in algoritmasında bir araç olarak kullandık.

YALANCIKOD (PSEUDOCODES)

```
Construct tree I_i.
For i from 1 to m — 1 do
begin [phase i + 1}
    For j from 1 to i + 1
            begin {extension j}
            Find the end of the path from the root labeled S[j..i] in the
            current tree, If needed, extend that path by adding character S[i + 1),
            thus assuring that string S[j. .i + 1] is in the tree.
            end;
end;
```

```
Construct tree T_1;  j_1 = 1;
for i = 1 to n – 1 do
begin {phase i+1}
   Do all implicit extensions.
   for j = j_i +1 to i +1 do
      begin {extension j}
         In the current tree find the end of the path from the root labeled
         t[j ... i]. If necessary, extend that path by adding character t[i+1],
         thus ensuring that string t[j...i+1] is in the tree.
         j_{i+1} := j;
         if rule 3 was applied then j_{i+1} := j – 1 and phase i+1 ends;
      end;
end;
```

Fig. 5. Ukk için genişletilmiş yalancıkod

.

Phase 1:
Scan the *String* and partition *Suffixes* based on the first *prefixlen* symbols of each suffix
Phase 2: Do for each partition:
1. START BuildSuffixTree
2. Populate *Suffixes* from current partition
3. Sort *Suffixes* on first symbol using *Temp*
4. Output branching and leaf nodes to the *Tree*
5. Push the nodes pointing to an unevaluated range onto the *Stack*

While *Stack* is not empty
6. Pop a node
7. Find the Longest Common Prefix (LCP) of all the suffixes in this range by checking the *String*
8. Sort the range in *Suffixes* on the first symbol using *Temp*
9. Write out branching nodes or leaf nodes to *Tree*
10. Push the nodes pointing to an unevaluated range onto the *Stack*

11. END

Fig. 6. Partition and Write Only TopDown

DENEYSEL SONUÇLAR

Program iki girdi alır: a) sonek ağacının oluşturulacağı katar ve b) girilen katarının n altkatarı. Kullanıcının girdileri girmesinin iki farklı yolu vardır. İlk yöntem, programın kendisinde aranacak dize ve alt dizeyi girmektir. İkinci yol, çalışma zamanı sırasında katarnin bulunduğu dosyanın adını ve altkatarı main fonkisyon parametresi olarak girmektir. Aşağıda şekil 7 ve şekil 8'de gösterildiği gibi, her ikisinin de çıktısı tamamen aynıdır.

Girdileri aldıktan sonra program aşağıdaki çıktıları verir:

- S katarı için bir sonek ağacı oluşturur.
- S sonek ağacının p altkatarın içerip içermediğini kontrol eder. içeriyorsa, altkatarın bulunduğu ilk pozisyonu gösterir ve altkatarın kac kere tekrar ettigini görüntüler.
- Sonek ağacı oluşturulan bir s katarı içinde tekrar eden en uzun altkatar bulur ve kaç kez tekrar ettigini gosterir.
- Sonek ağacı oluşturulan bir s katarı içinde en çok tekrar eden altkatarı bulur ve kaç kez tekrar ettigini goruntuler.

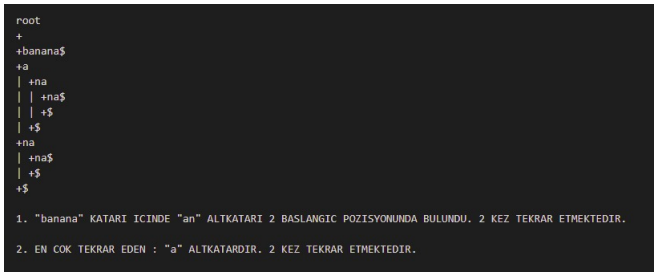

Fig. 7.

Fig. 8.

KAYNAKÇA

REFERENCES

[1] https://core.ac.uk/reader/82138620
[2] Algorithms on Strings, Trees, and SequencesDan GusfieldUniversity of California, DavisCambridge University Press1997
[3] https://tr.hrvwiki.net/wiki/Suffix_tree.
[4] http://web.stanford.edu/ mjkay/gusfield.pdf
[5] http://fragglet.github.io/c-algorithms/
[6] http://www.vilo.com/edu/2002-03/Software/Loeng5 _Suffix_Trees/Suffix_Trees /cs.haifa.ac.il/shlomo/suffix_tree/
[7] https://www.youtube.com/watch?v=3CbFFVHQrk4
[8] https://www.youtube.com/watch?v=N70NPX6xgsA
[9] https://visualgo.net/en/suffixtree
[10] https://www.tutorialspoint.com/cprogramming/c_ command_line_arguments.htm
[11] https://rosettacode.org/wiki/Suffix_tree
[12] https://stackoverflow.com/questions/737257/how-to-convert-c-code-to-c
[13] https://www.geeksforgeeks.org/enumeration-enum-c/
[14] https://www.youtube.com/watch?v=qh2leThTv0Y
[15] Book Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology by Dan Gusfield
[16] https://developerinsider.co/graphics-graphics-h-c-programming/
[17] https://www.geeksforgeeks.org/represent-tree-using-graphics-in-c-c/
[18] https://www.sanfoundry.com/cpp-program-implement-suffix-tree/
[19] https://en.wikipedia.org/wiki/Longest_repeate d_substring_problem
[20] https://dmcconnell-comp150.herokuapp.com/suffix _trees/construction
[21] https://stackoverflow.com/questions/3306279/tries-and-suffix-trees- implementation
[22] https://stringfixer.com/tr/Suffix_tree
[23] https://caetanogenete.github.io/Visualise-Suffix-Tree/
[24] https://www.overleaf.com/