Excel Dosyalarındaki Verilere Dayalı Soy Ağacı Oluşturma

Sezer Yildirim 210202001@kocaeli.edu.tr

Kasım 2022

1 Özet

Proje amacı excel dosyalarındaki verilerin okunarak bu veriler üzerinde soy ağacı oluşturulup verilerin ortak noktaları üzerinde sıralama, sınıflandırma gibi işlemlerin uygulanmasıdır. Bu çalışma class yapıları ve overriding gibi işlemler barındırmasıyla nesneye yönelik programlama özelliği taşımaktadır. Program java üzerinde yapılmış olup standart kütüphaneler dışında bir kütüphane kullanılmamıştır.

2 Giriş

İlk etapta projenin temel amacı olarak da belirtilen soy ağacının oluşturulması hedeflenmiştir. Projenin oluşturulması ve genel algoritmik mimarinin kurulmasında örnekler incelenmiş ve soyağacı gibi yapılarda sıralama, dizi oluşturma, bağlı dizi ve rekürsif yapıların oldukça kullanıldığı görülmüştür. Dolayısıyla projede de bu yapılar kullanılmıştır. Projenin excel verilerini okuyabilmesi, saklaması ve işlenebilmesi amacıyla birbiriyle kesişimlerinin anlaşılması gerekmektedir.

3 Program Mimarisi

Proje NetBeans IDE'si üzerinde java kullanılarak gerçekleştirilmiştir. Proje isteri olarak kişi için bir sınıf oluşturulması gerekmektedir. Bu sınıfın özellikleri excel de karşılık gelen tc no (id), ad, soyadı, doğum tarihi, anne adı, baba adı, kan grubu ve meslek özellikleri, kızlık soyadı ve cinsiyet gibi özellikleri karşılayabilir olmalıdır.

class Person private String id, name, surname,
motherName, fatherName, bloodGroup,
job, maidenName, gender, birthdate,
maritalStatus, partnerName;

Excel satırlarının her biri bir düğümü karşılık gelecek şekilde yapılandırılmalıdır. İlk düğümde bulunan yapı dışında geri kalan tüm verilerin anne baba ilişkisi vardır. Bu düğümlerin yapılabilmesi amacıyla Personları "point" edebilecek bağlı liste mantığına dayalı bir Arraylist oluşturulmuştur.

class Node
private Person person;
private Node Father, Mother;
private ArrayList <Node> childrens = new ArrayList<Node>();

Grands için ayrıca bir Node dizisi oluşturulmuştur. Bu şekilde yapılma sebebi ana dizide 2 for döngüsü ile yapılacak bir arama işlemi yerine kod karmaşıklığını indirgeyerek tek bir döngüde yapılabilmesini sağlamaktır.
public Node getGrandByName(String name)
for (int i = 0; i ¡ grands.size(); i++)

```
if (
grands.get(i).getPerson().getName().
equals(name))
return grands.get(i);


return null;
```

Burada ağacın oluşturulma aşamasını sağlayan aksiyon aşamalarının sağlandığı Tree classı'nın constructor yapısı kullanılarak Person class'ı ile oluşturulan excel verilerinin bağlı yapı oluşturması sağlanır.

```
public Tree(ArrayList¡Person¿ persons)
for (int i = 0; i ¡ persons.size(); i++)
nodes.add(new Node(persons.get(i)));
```

Son olarak print aşamasında yukarıda giriş kısmında bahsedildiği gibi recursive yapı kullanılmıştır. Recursive fonksiyon, içerisinde kendini çağıran fonksiyondur. Özyineleme yöntemiyle karmaşık problemlerin çözümü, döngülerle yapılan çözümlerden daha basit ve daha kısadır. Özyineleme yöntemi, karmaşık problemi taban denilen en basit haline indirger, oradan başlayarak adım adım asıl problemi çözer. Soy ağacı yapısı bu duruma çok uygun olduğundan yazdırma aşamasında recursive tercih edilmiştir.

```
public void printTree()
for (int i = 0; i ¡ grands.size(); i++)
if (grands.get(i).getPerson().
getGender().equals("Erkek"))
printTree(grands.get(i), 0);

  private void printTree(Node node, int level)
System.out.println();
for (int i = 0; i ¡ level; i++)
System.out.print("——->");

System.out.println(node.getPerson().toString()
+ " - " +
node.getPerson().getPartnerName());
for (int i = 0; i ¡ node.getChildrens().size();
i++)
if (node.getChildrens().size() == 0)
break;
printTree(node.getChildrens().get(i), level +
1);
```

4 Soyağacı Üzerinde Gerçekleştirilen Fonksiyonlar

Temel hedef olan soy ağacı yapısı tamamlandıktan sonra proje isterleri olarak 9 adet soyağacı üzerinde gerçekleştirilmesi beklenen görev ve son madde olarak karmaşıklık hesabı istenmektedir.
1. isterde "Depth First" algoritması bir başlangıç noktası belirleyip bu noktadan (node) komşu node'lara doğru gezilir ancak tek seferde sadece bir adet komşuya bakılır. Bu sayede bakılan her komşu node yığına eklenir ve bir önceki node yığından çıkar. Ancak projede tam olarak depth first algoritması kullanılarak bir arama gerçekleştirilmemiştir. Bu aşamada çocuğu olmayanların bulunması gerçekleştirilirken diziye kaydedilme işlemi bir HashSet<Node> yapısı ile gerçekleştirilmiştir. ilk ister olan bu yapı birkaç özellik içerdiğinden kod parçası aşağıdaki gibi paylaşmıştır. Geri kalan yapılarda bulunan isterler 1. maddenin daha basit haliyle çözümlendiğinden paylaşılmasına gerek görülmemiştir.

```
  public void findPersonWithoutChildrens()
```

for (int i = 0; i ¡ grands.size(); i++)
findPersonWithoutChildrens(grands.get(i));
System.out.print("Cocugu olmayan kisiler:");
for (Node node : childless)
System.out.print(" " +
node.getPerson().toString() + " - ");
System.out.println();

private static HashSet<Node> childless =
new HashSet<Node>();

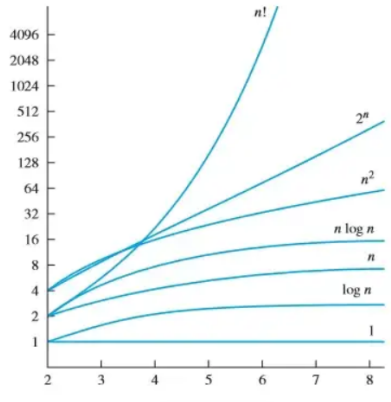
private void findPersonWithoutChildrens(Node node)
if (node.getChildrens().size() == 0)
childless.add(node);
for (int i = 0; i ¡ node.getChildrens().size();
i++)
findPersonWithoutChildrens(node.
getChildrens().
get(i));

1. isteri sağlayan findPersonWithoutChildrens() metodu incelendiğinde iki metot olarak görülmektedir. Bu metotlardan ilk tanımlanan metot üzerine overload işlemi uygulanmıştır. Aynı şekilde recursive yapı burada da kullanılmıştır. Kullanılması istenilen metot ilk metot olması istendiğinden ikinci metot private olarak belirlenmiştir.

5 Algoritma Analizi ve BIG-O Notasyonu

Bilgisayar bilimlerinde iki temel kaynak bulunur. Bunlar yer(hafıza) ve zaman(hız) dır.

Bir bilgisayar programı, ne kadar fazla yer kaplıyor ve ne kadar yavaş çalışıyorsa o kadar kötü demektir. Hızlı çalışan ve az yer kaplayan program ise daha iyi olarak düşünülebilir.
Big(O) algoritmaya girdi olarak verilen verilerin miktarına bağlı olarak sonuçlar üretilir. Elde edilen bu sonuçlar ise ilgili algoritmanın karmaşıklığı olarak tanımlanır.



Yukarıdaki fotoğrafta algoritma karmaşıklığına göre büyüme hızları görülmektedir. Bir işlemi n olarak aldığımızda bu işlemin sayısına bağlı olarak görseldeki değerler yakalanabilir. Her bir fonksiyon için incelendiğinde:

findpersonwithoutchildrens(400) $2n^2$
$stepsisterbrother(424)n^2$
$sameblood(447)n$
$samejob(466)n^3$
$samename(490)n^2$
$relation(503)5n$
$createtreeforperson(562)2n^2$
$maxdepth(589)n^2$
$generations(614)n + n^3$

6 Referanslar

1) medium.com
https://seymagoksel.medium.com/algoritma-analizi-ve-big-o-notasyonu/(17.12.2022)
2) github.com (12.12.2022)
3java.com.tr
https://www.java.com/tr/(18.12.2022)

7 UML Diagram