YILDIZ TEKNİK ÜNİVERSİTESİ
ELEKTRİK-ELEKTRONİK FAKÜLTESİ
BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

BLM 3021 ALGORİTMA ANALİZİ

ÖDEV 4

Muhammet Ali Şen - 20011701

1. YÖNTEM

Ödevde kabaca bir sosyal medya uygulamasındaki bot hesapların tespiti ile o sosyal ağdaki 'influencer' olarak tabir edilen etkili kullanıcıların bulunması istenmiştir. Sosyal ağımıza ait veriler 'socialNET.txt' dosyasının içerisinde bulunmaktadır. Bu dosyada kullanıcı hesaplarının id, isim, soyisim ve takip ettiği diğer kullanıcıların id bilgileri yer almaktadır.

'socialNET.txt' dosyasında bazı satırlarda parametreler ',' (virgül) ile ayrılırken bazılarına ',' (virgül) konulmasının unutulduğu görülmüştür. Bu nedenle eksik görülen yerlere ',' (virgül) konularak düzenlenmiştir.

İlk dosya okunması sonrası kaç kullanıcı olduğu tespit edilmiştir. 12 kullanıcının tespiti sonrası 12x12 lik bir komşuluk matrisi oluşturulmuş ve ayrıca kullanıcıların bilgilerinin tutulduğu USER isimli struct yapımız içinde 12 elemanlık bir struct dizisi açılmıştır.

Bu işlem sonrasında ödevde istenen eleme adımları uygulanarak sürekli olarak komşuluk matrisi güncellenmiş ve komşuluk matrisi üzerinden de USER struct dizimiz güncellenerek işlem yapılmıştır. En sonda ise kalan değerler ekrana gönderilmiştir.

2. UYGULAMA

Yukarıda sayılı işlem adımları için bir dizi fonksiyon tanımlanmıştır. Bu fonksiyonların karmaşıklıkları aşağıda hesaplanmıştır. Belirtmek gerekir ki tüm fonksiyonlarda zaman karmaşıklığı hesabı için döngüler içindeki 'Comparison' karşılaştırmalar esas alınmıştır. Basic operator olarak for döngüleri içindeki karşılaştırmalar tespit edilmiştir. Tüm fonksiyonlara kabaca değinmek gerekirse;

- **countUserId** ve **addUser2Array** fonksiyonunun zaman karmaşıklığı O(n)'dir. Burada n, dosyadaki satır sayısıdır. Basic Operation işlemi fgets fonksiyonuyla okunan satırdır.

  Fonksiyon, dosyadan bir seferde en fazla 100 karakter okuyan fgets foksiyonunu kullanarak dosyanın her satırını okur. Fonksiyon daha sonra her satırı tokenize yaparak ve kullanıcı kimliğini çıkararak işler. Her satırın işlenmesi sabit bir süre alır, bu nedenle fonksiyonun toplam zaman karmaşıklığı, dosyadaki satır sayısı açısından doğrusaldır.

  Fonksiyonun sonundaki fseek işlevi, dosya işaretçisinin konumunu sabit bir süre alan dosyanın başlangıcına sıfırlar. Bu nedenle, fonksiyonun genel zaman karmaşıklığını etkilemez.

- **beingAdjMatrix** fonksiyonunun zaman karmaşıklığı O(n^2) şeklindedir; burada n, kullanıcı sayısıdır. Kullanıcı sayısınca açılan matris içerisinde inDegree değerleri 1 olarak işaretlenmiştir. Matris gezme işlemi kabaca O(n^2) şeklinde gerçekleşir.

- **createFollowingsFollowers** fonksiyonunun zaman karmaşıklığı, n'nin kullanıcı sayısı olduğu O(n^2) şeklindedir. Fonksiyon, komşuluk matrisi üzerinde yinelenir ve matristeki girişlere göre her kullanıcının aşağıdakileri ve takipçi listelerini günceller.

Fonksiyon, gerektiğinde aşağıdakiler ve takipçiler listeleri için ek bellek ayırmak için realloc fonksiyonunu kullanır. Realloc fonksiyonu zaman karmaşıklığı uygulamaya bağlıdır, ancak genel olarak n öğe için O(n)'dir, dolayısıyla işlevin genel zaman karmaşıklığını etkilemez.

- **eliminateMatrix** fonksiyonunun zaman karmaşıklığı O(n^2) olacaktır.İlk döngü, "while(check != 0)" döngüsü içinde olup, bu döngü maksimum n kere dönecektir.İkinci döngü, "for(i=1; i<allUser+1; i++)" döngüsü içinde olup, bu döngü maksimum n kere dönecektir.

  Üçüncü döngü, "for(j=1; j<allUser+1; j++)" döngüsü içinde olup, bu döngü maksimum n kere dönecektir. Ayrica içinde yer alan Dördüncü döngü, "for(j=1; j<allUser+1; j++)" döngüsü içinde olup, bu döngü de maksimum n kere dönecektir. Bu nedenle, bu fonksiyonun zaman karmaşıklığı O(n^2) olacaktır.

- **updateFollowingsFollowersByMatrix** fonksiyonunun zaman karmaşıklığı O(n^2) olacaktır.Fonksiyon içerisinde iki ayrı For dönüsü ile tüm matris tablosu gezilmekte ve düğümler güncellenmektedir. Iki for güngüsü de "for(i=1; i<allUser+1; i++)" maksimum n kere dönecektir. Bu nedenle, bu fonksiyonun zaman karmaşıklığı O(n^2) olacaktır.

- **calculateAlltotalFollowers** fonksiyonunun zaman karmaşıklığı O(n^2) olacaktır.Fonksiyon içerisinde iki ayrı For dönüsü ile tüm matris tablosu gezilmekte ve visited dizisi oluşturulmaktaıdr. İki for döngüsü de "for(i=1; i<allUser+1; i++)" maksimum n kere dönecektir. Bu nedenle, bu fonksiyonun zaman karmaşıklığı O(n^2) olacaktır.

- **calculatetotalFollowersDFS** fonksiyonunun zaman karmaşıklığı O(V + E) olacaktır, burada V, kullanıcı dizisinde bulunan kullanıcı sayısını ve E, ilişki matrisinde bulunan tüm ilişki sayısını temsil eder.

  Fonksiyon, kullanıcı dizisi içinde verilen kullanıcının takipçi listesinde dolaşır ve tüm takipçilerin takipçi listesi için yine DFS (Depth First Search - Derin Önce Arama) algoritmasını kullanır. Bu nedenle, her kullanıcı için en fazla bir kez ziyaret edilecektir ve bu nedenle V sayısı kadar zaman alır. Her ilişki için de bir kez ziyaret edilecektir ve bu nedenle E sayısı kadar zaman alır. Bu nedenle, toplam zaman karmaşıklığı O(V + E) olacaktır.

- **printByFollowers** ve **printUserArray** fonksiyonlarının zaman karmaşıklığı O(n) olur. Fonksiyon USER veri yapımızdaki düğümleri teker teker dolaşarak eliminate sonrası kalanlardan parametrelere uygun olanları ekrana yazar. Kullanıcı dizisi üzerinden ulaşıldığı için tek döngü içerisinde toplam zaman karmaşıklığı O(n) olacaktır.
- **printAdjMatrix** fonksiyonu yine iç içe iki ayrı for döngüsü nedeniyle her döngüde n karmaşıklığına sahip olmasından dolayı O(n^2) karmaşıklığındadır.

Dosya okuması sonrası yapılan işlemler sırasıyla;

1. Kullanıcı sayısı tespit edilerek komşuluk matrisimiz bu kullanıcı sayısına uygun olarak oluşturulmuştur.

2. Yine kullanıcı sayısı doğrultusunda USER isimli struct veri yapımızdan oluşan bir dizi dinamik olarak oluşturulmuştur.

   USER isimli veri yapımız aşağıdaki şeklindenir.

```
typedef struct user
{
   int id;
   char *name;
   char *surname;
   int *followings;
   int *followers;
   int followingsCount;
   int followersCount;
   int *totalFollowers;
   int totalFollowersCount;
}USER;
```

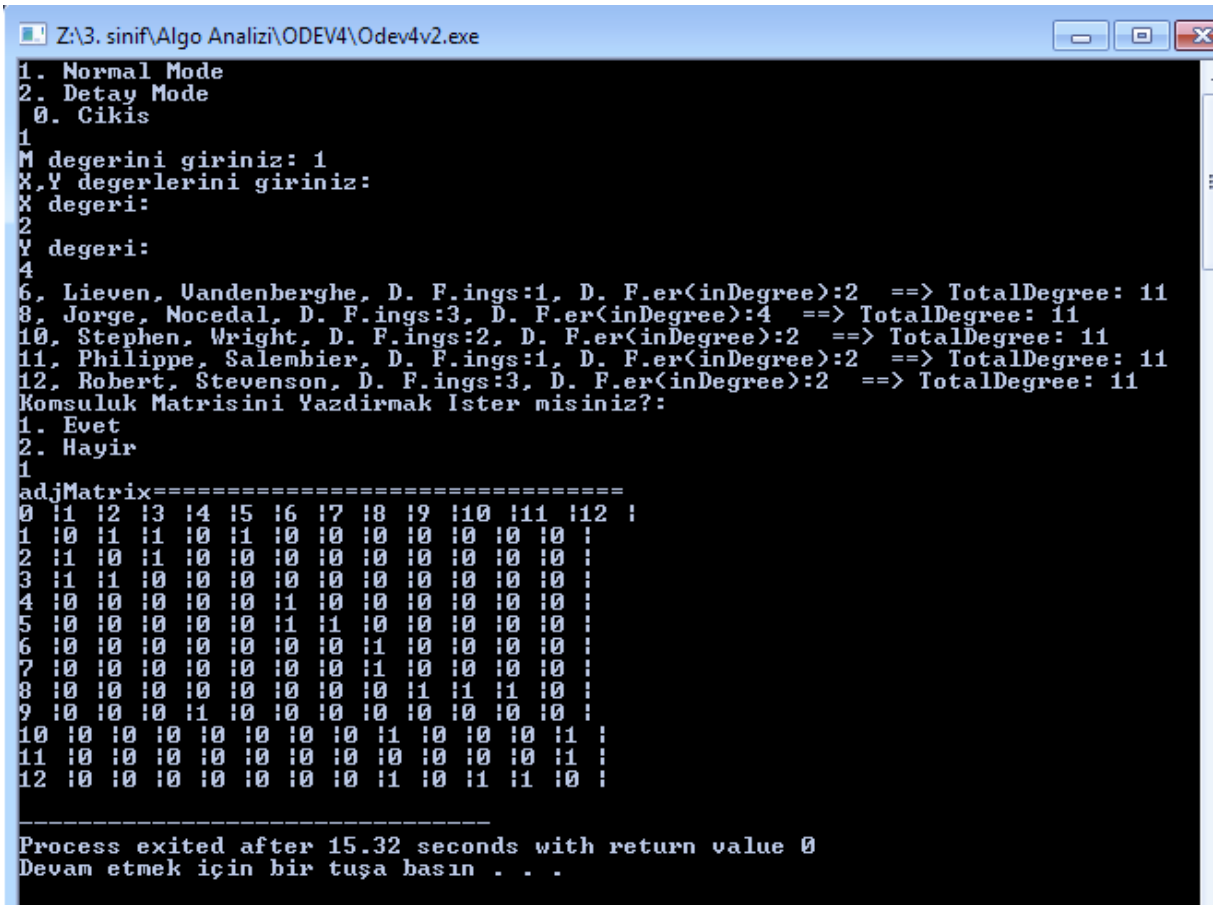
   Burada belirtilen hesabın kimleri takip ettiği followings listesine dinamik olarak eklenmiş ve followingsCount ile takipçi sayısı tespit edilmiştir.

   Bahse konu hesabın kimler tarafından takip edildiği ise followers listesine dinamik olarak eklenmiştir. Bu liste ödevde belirtilen '**InDegree'** tanımındadır. Bu listenin eleman sayısı ise followersCount değişkenindedir.

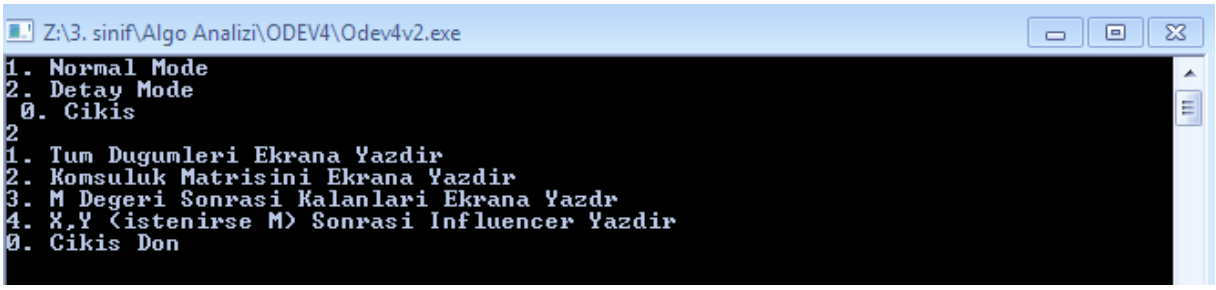
   Kendisini takip eden (followers) doğrudan veya dolaylı olan tüm inDegree'lerin eleman sayısı totalFollowersCount değişkeninde saklanırken bu elemanlar ise totalFollowers a dinamik olarak eklenmektedir.
3. Oluşturulan komşuluk matrisi sonrası kullanıcıdan alınan M, ve X değerleri doğrultusunda eleme işlemleri **eliminateMatrix** fonksiyonunda komuşuluk matrisi üzerinden yapılmıştır. Bu işlemler ile komşuluk matrisi sürekli olarak güncellenmiştir.
4. Güncellenen komşuluk matrisi sonrası USER dizimiz de komşuluk matrisi değerlerine uygun olarak yeniden düzenlenmiştir. Bunun için **updateFollowingsFollowersByMatrix** fonksiyonu çalıştırılmıştır.
5. Bu güncelleme ile Y değerine uygun olarak doğrudan veya dolaylı tüm inDegree'lerin hesaplanması işlemi için DFS algoritması kullanılmıştır. Bunun için öncelikle **calculateAlltotalFollowers** fonksiyonu çağırılmış ve visited dizisi oluşturulmuş ve bu fonksiyon içerisinde recursive olan **calculatetotalFollowersDFS** fonksiyonu çağırılmıştır.
6. **calculatetotalFollowersDFS** fonksiyonu recursive olarak her bir id değerini takip eden hesapları stack veri yapısına atmış ve stack'e atılan id nin takipçileri de stack'e atılmak suretiyle özyinelemeli şekilde çalışmaktadır. En sonunda stack'e atılacak id değerinin kalmaması veya başlangıç id değerine ulaşılması halinde stack de kalanlar doğrudan veya dolaylı takipçileri oluşturmaktadır.
7. Bu işlemler sonunda 3 ayrı print fonksiyonu yazılmıştır. Belirtilen print fonksiyonlarında bazı satırlar yoruma alınmıştır. İstenirse bu satırlar yorumdan çıkarılarak tüm hesapların doğrudan takipçilerinin id değerleri, doğrudan takip ettiği id değerleri ve doğrudan veya dolaylı takipçilerinin id değerleri de görülebilir.

3. SONUÇ

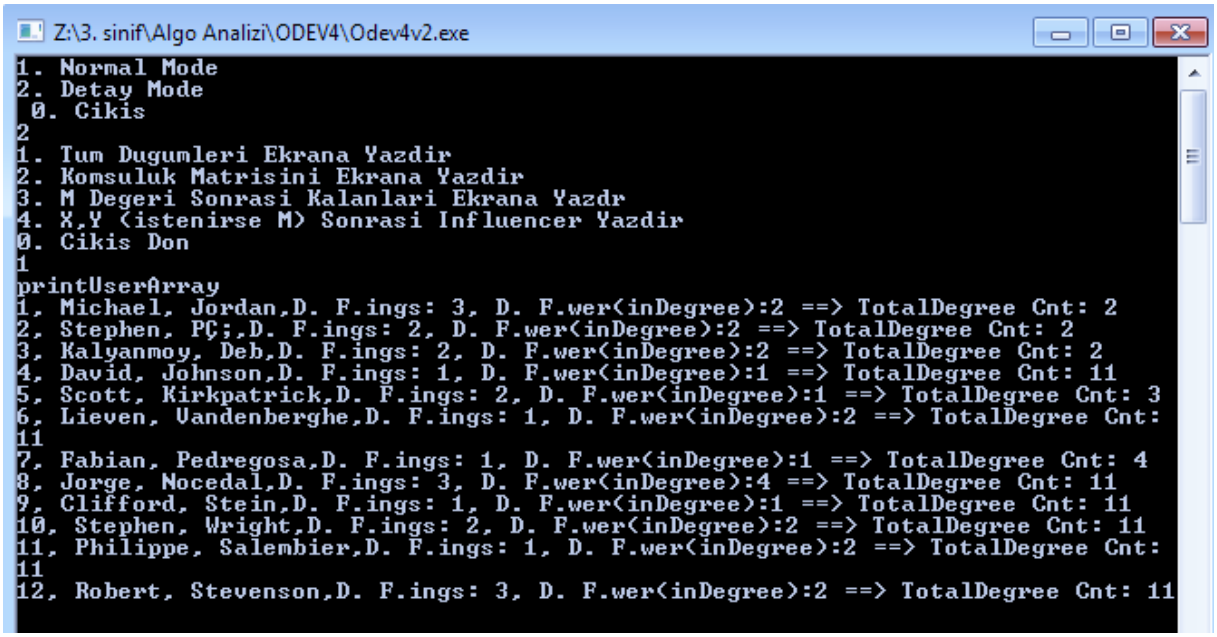
Program istenildiği şekilde Normal ve Detay mod olarak çalıştırılabilmekteedir. Normal modda istenilen M, X ve Y değerleri sonrası kalan hesapların bilgileri ve istenirse komşuluk matrisi ekrana yazdırılmaktır. Normal moda ait ekran görüntüleri;
M: followers(inDegree) değeri m sayısının altında olanları hepsi m üzerinde kalana dek elemek içindir.
X: X>M şartıyla doğrudan inDegree değeri bir x sayısından büyükleri
Y: Doğrudan veya dolaylı inDegree değeri bir y sayısından büyükleri



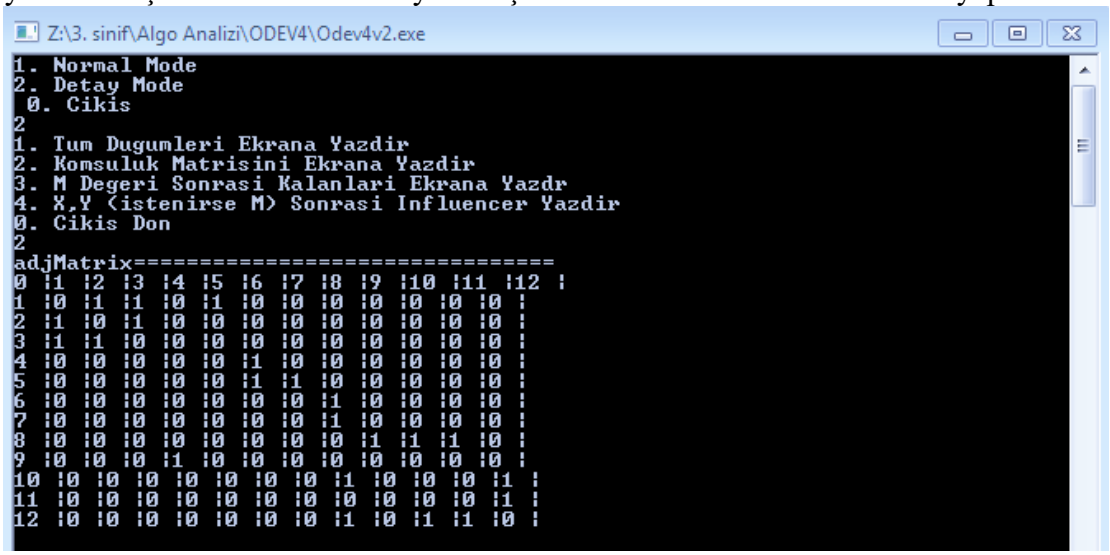
M:1 X:2 Y:4 değeri için kalan hesap İD değerleri : 6,8,10,11,12 olmuştur.
Detay Modda ise 4 başlık mevcuttur.



1. Tüm düğümlerin başlangıç değerlerinin (dosya okunması sonrası) ekrana yazılması,



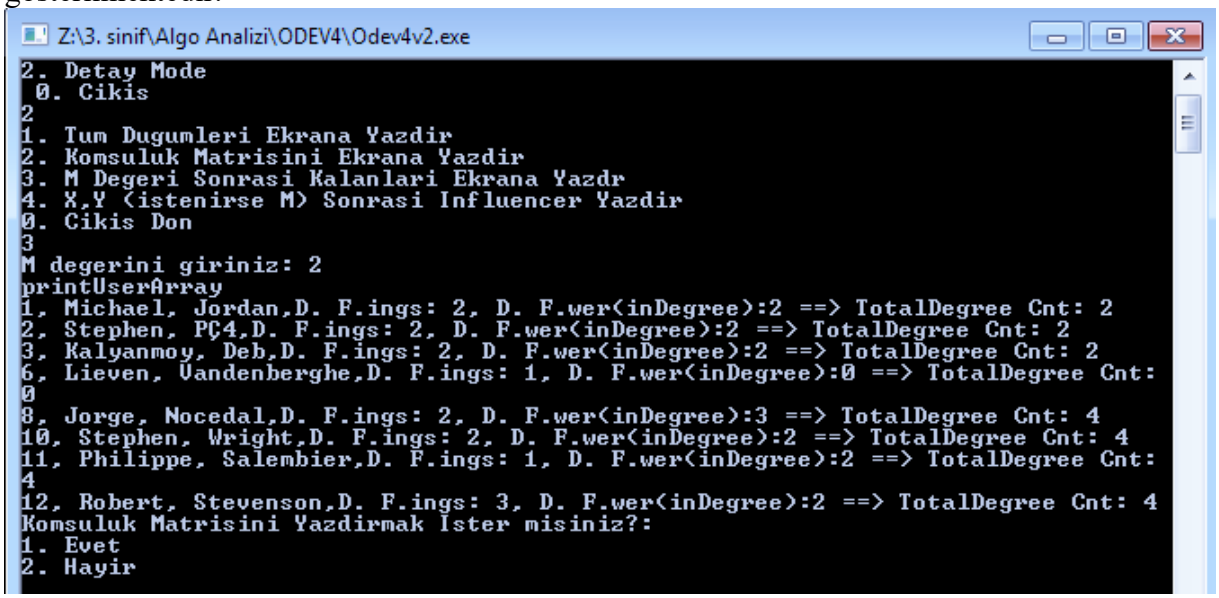
2. Komşuluk matrisinin başlangıç değerlerinin (dosya okunması sonrası) ekrana yazılması işlemi vardır. Ekrana yazım işlemis sonrasında iki farklı okuma yapılabilir.



A. ‘Satırlardaki id değerlerine karşılık gelen 1 olan sütunlardaki id değerlerini takip ettiği’ şeklinde okunabilir.

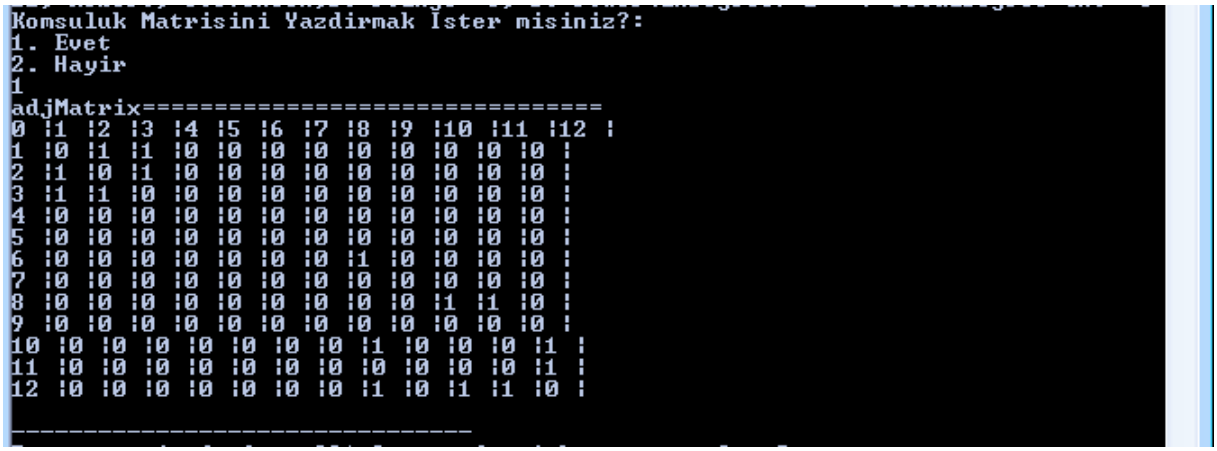
B. ‘Sütunlardaki id değerine karışık gelen 1 olan satırlardaki id değeri tarafından takip edildiği’ (inDegree) şeklinde de okunabilir.

3. M değeri sonrasına Kalanlar (X ve Y değeri alınmadan) ekranda gösterilmektedir.

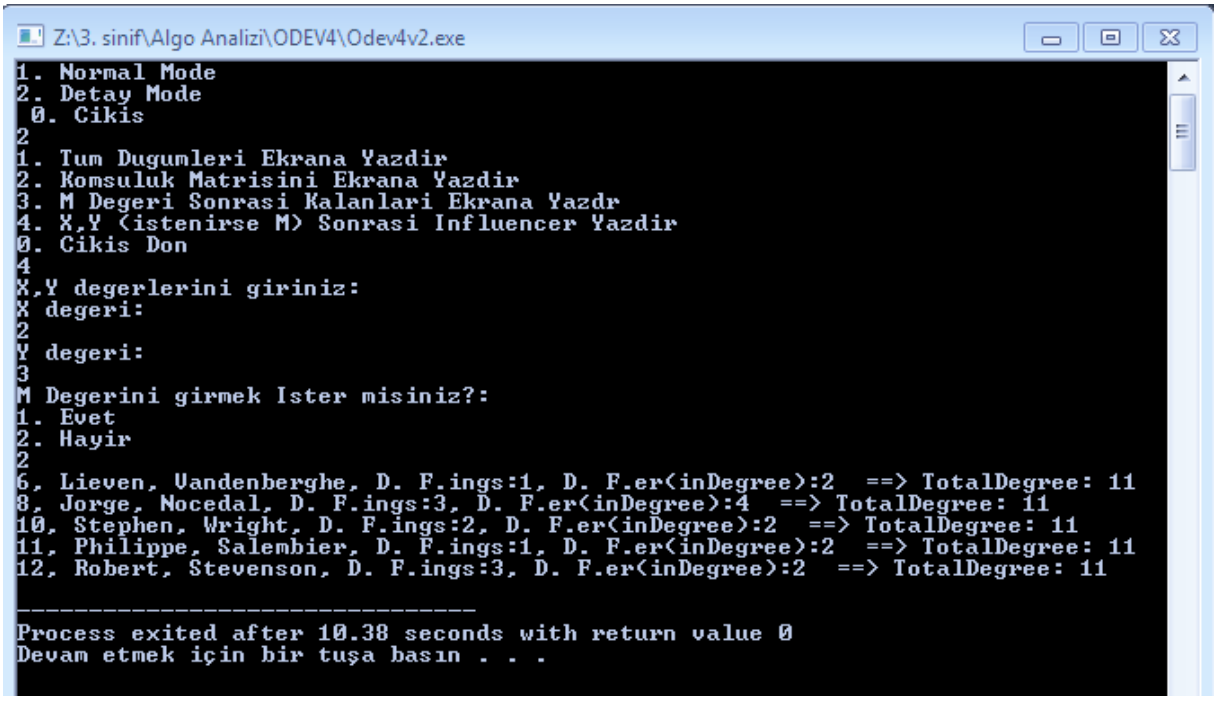


M:2 değeri sonrası kalanlar.

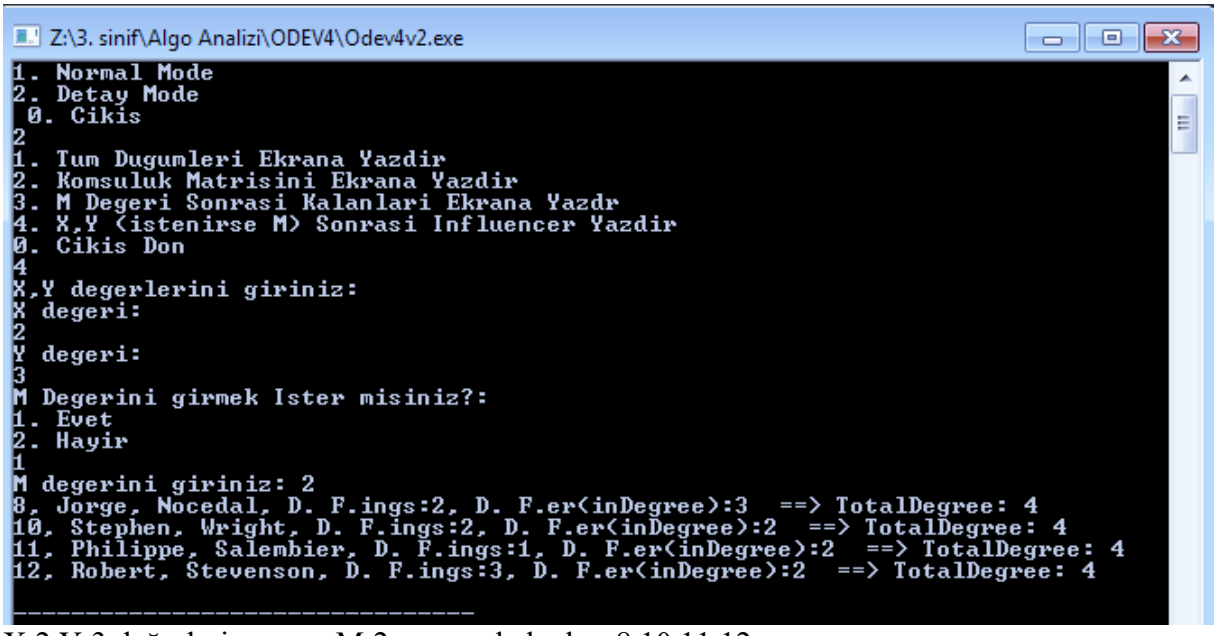
Bu işlem sonrası istenirse geri kalanları gösteren komşuluk matrisi de ekrana yazdırılabilir.



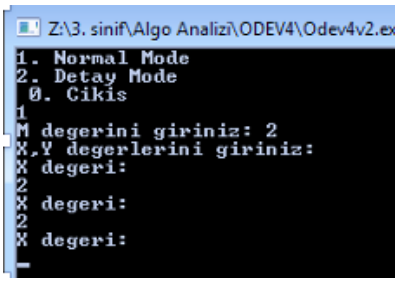
4. X ve Y (eğer istenirse M) değeri sonrası oluşan durumu ekrana yazdırılması,



X:2 Y:3 değerleri sonrası M değeri alınmadan kalanlar: 6,8,10,11,12

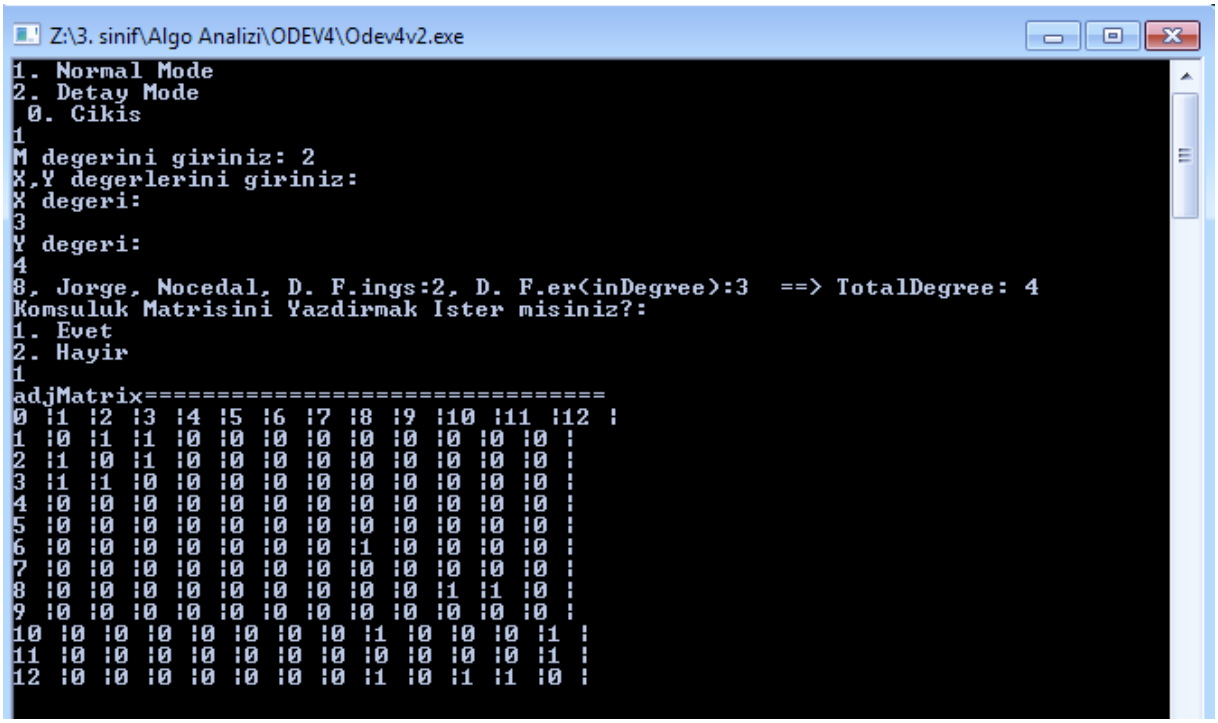


X:2 Y:3 değerleri sonrası M:2 sonrası kalanlar: 8,10,11,12



Son olarak Detay(2) modunun 4. Menüsü haricinde (çünkü önce X sonra M alınmaktadır) tüm X > M şartı sağlanana dek yeniden eleman alması gereklidir.

İnfluencer tespiti için en iyi sonuçlar M:2 X:3 Y:4 değerlerinde oluşmaktadır.



M:2 X:3 Y:4 için influencer: 8 kalmıştır.