**Graf Yapısı**

Bir veri yapısı türüdür. Düğümlerden ve bu düğümleri birbirine bağlayan linklerden oluşur.

taslak, çizim, daire, kırpıntı çizim içeren bir resim

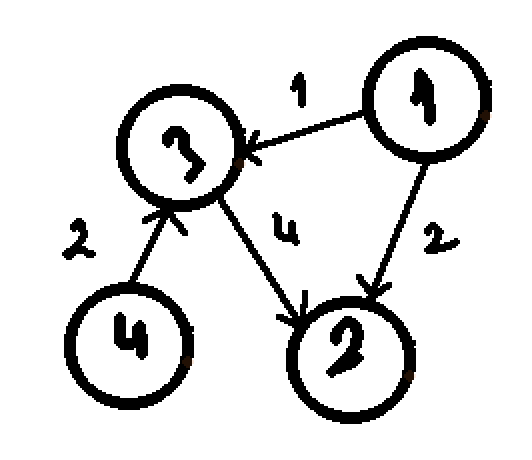
Açıklama otomatik olarak oluşturuldu

Örneğin yukarıdaki şekilde içinde sayı bulunan yuvarlaklar düğümleri temsil ederken aradaki çizgiler linkleri yani bağlantıları temsil eder.

daire, çizim, taslak, kırpıntı çizim içeren bir resim

Açıklama otomatik olarak oluşturuldu

Linkler yönlü veyahut yönsüz olabilirler. Bunun doğrultusunda graflar da yönlü veya yönsüz olacak şekilde adlandırılabilir. Sağ tarafta bulunan şekilde yönlü bir graf örneği görüyoruz. Okların yönü bağlantıların yönünü temsil ediyor.



Ayrıca linkler ağırlıklı veya ağırlıksız olabilir. Buna göre graf ağırlıklı veya ağırlıksız olarak isimlendirilebilir. Sağ yukarıda bulunan görsele ağırlıklar ekleyerek sağda bulunan ağırlıklı grafı elde edebiliriz.

Grafların “path graph”, “planar graph”, “cycle graph” gibi farklı türleri bulunmaktadır. Projenin anlaşılması doğrultusunda bipartite graflara değinmek yardımcı olacaktır.

**Bipartite (İki Parçalı) Graf**

Düğümlerin iki parçaya ayrıldığı ve ne ilk gruptaki ne de ikinci gruptaki düğümlerin kendi aralarında link bulundurmadığı graflara bipartite graf denir.

daire, diyagram, çizgi, çizim içeren bir resim

Açıklama otomatik olarak oluşturuldu

Soldaki şekilde bir bipartite graf örneği görüyoruz. Siyah daireler birinci grubu temsil ederken mor daireler ikinci grubu temsil ediyor. Görülebileceği üzere ne siyah daireler ne de mor daireler kendi aralarında bir bağ içermiyorlar. Ayrıca linklerin yönünün bulunmadığını göz önünde bulundurarak bunun yönsüz bir graf olduğunu da söyleyebiliriz.

**Grafların Kodlanması**

Graflar kodlanırken iki boyutlu array, array + linked list veya linked list + linked list gibi farklı yaklaşımlar sergilenebilir. Raporun kapsamı göz önünde bulundurulduğunda projede kullanılan array + linked list sisteminden bahsetmek zannediyorum yeterli olacaktır.

**daire, ekran görüntüsü, diyagram içeren bir resim

Açıklama otomatik olarak oluşturulduArray + linked list yapısı** tüm düğümlerin bulunduğu bir arrayden oluşur ve bu array’in her bir elemanının bir linked list’in başlangıcı olduğu bir yapıdır. Linked listler grafta düğümün bağlı olduğu diğer düğümleri içerir, ayrıyeten eğer linkler ağırlıklıysa linked list’in elemanlarına bu ağırlıklar eklenebilir zira projede de böyle yapılmıştır.

Bir önceki sayfada verilen graf yapısı array + linked list kodlamasında yukarıdaki gibi temsil edilebilir. Soldaki uzun dikdörtgen tüm düğümleri içeren bir arrayi temsil ediyor, oklarla ilerleyerek de bu düğümlerin bağlı olduğu diğer düğümlere erişebiliyoruz.

**Proje Hakkında Genel Bilgiler**

**Amaç:** Çeşitli algoritmalar vasıtasıyla elimizdeki verilerden yola çıkarak film önerisinde bulunmak.

Projenin kodu C programlama dilinde yazılmıştır.

**KODUN İÇERİĞİ**

Kod temel olarak “example.txt” adlı, verileri bulunduran dosyanın okunduğu ve bu doğrultuda graf yapısının oluşturulduğu, 6 farklı algoritma için fonksiyonların bulunduğu, programı çalıştıran kişiden id ve öneri sayısının alındığı ve bu doğrultuda tüm algoritmaların çalıştırılarak sırayla ekrana basıldığı kısımlardan oluşmaktadır.

“example.txt” sırasıyla userid, filmid, rating ve kullanılmayan bir diğer sütunlarından oluşmaktadır.

**Graf Yapısının Oluşturulması**

daire, diyagram, çizim, çizgi içeren bir resim

Açıklama otomatik olarak oluşturuldu

Bipartite graf başlığı altında bulunan görsele ağırlıklar ekleyerek projede kullanılana benzer bir graf elde edebiliriz. Siyah dairelerin kullanıcıları, mor dairelerin ise filmleri temsil etmektedir. Ağırlıklar 1’le 5 arasında değer almaktadır. (4. Algoritma için ağırlıkta yapılan değişiklik daha sonra açıklanacaktır.)

Graf yapısı array + linked list kullanılarak kodlanmıştır.

**Öneri sistemleri**

Koddaki sıraya göre:

1. **RandomRec:** Kullanıcının izlemediği filmler arasından rastgele filmler önerir.
2. **MostWatchedRec:** En çok kullanıcı tarafından izlenmiş filmleri önerir.
3. **SimilarUserReccom:** Aldığı kullanıcı için en fazla ortak film izlediği kullanıcıyı bulur ve bu en benzer kullanıcının en yüksek puan verdiği filmleri önerir.
4. **ShortestWeighPathRec:** En yakın ağırlıklı uzaklıklı filmi önerir. Puanları ağırlık kabul ettiğimizde önereceği öncelikli filmler beğenmediği filmleri beğenmeyen kullanıcıların beğenmediği diğer filmler olduğu için bu fonksiyon ve yardımcı fonksiyonlarda ağırlık 6 – verilen\_puan kabul edilmiştir.
5. **closeAverageRec:** Kullanıcının izlediği en popüler 5 filmi bulur ve bu filmlere verdiği ortalama puanı hesaplar. Daha sonra bu 5 filmin tamamını izlemiş diğer kullanıcıları bulur ve bu kullanıcıların da 5 filme verdiği ortalama puanı hesaplar. Ortalama puanı alınan kullanıcıya en yakın olan kullanıcı benzer kullanıcı olarak belirlenir ve sonrasında SimilarUserReccom gibi bulunan benzer kullanıcının en yüksek puan verdiği filmler önerilir.
6. **bestRatingRec:** Bir filmi izleyen tüm kullanıcıların bu filme verdiği puanın ortalaması alınarak filmin puanı hesaplanır. En yüksek puanlı filmler önerilir. (10’dan az kişi tarafından izlenmiş filmler öneriye dahil edilmez.)

Not: Bütün sistemlerde kullanıcının izlemediği filmler arasından öneri yapılmıştır.

**Karmaşıklık**

(u->kullanıcı sayısı, f->film sayısı).

WatchedFilms: Arrayin bir düğümünden başlayarak bu düğümün başı olduğu bağlı listenin sonuna kadar gittiğinden dolayı O(bağlı listenin eleman sayısı) diyebiliriz. Bu sayının da filmler için maksimum kullanıcı sayısı olduğunu göz önünde bulundurarak en kötü durum için karmaşıklığına O(f) diyebiliriz.

WatchedFilms2: İçerdiği selection sort’taki iç içe while döngülerinden dolayı en kötü durumdaki karmaşıklık O[f^2] olur.

**randomRec:** Her seferinde rastgele bir sayı bulup izlendi mi diye kontrol ettiği için en kötü durumda sonsuza kadar çalışır ama bu pek olası değil. Kullanıcının izlediği film sayısının toplam film sayısına bölümüyle oranının arttıkça çalışma süresinin artacağını söyleyebiliriz.

Ortalama Çalışma Süresi: 0.000 saniye

watchedBy: En kötü durumda O(u) karmaşıklığına sahiptir.

**mostWatchedRec:** İçerdiği quick sort’a tüm filmleri içeren bir array verildiğinden dolayı quick sort’un karmaşıklığı en kötü durumda O(u^2)’dir. [Ortalama durumda ise O(log2u).]

watchedBy fonksiyonunun tüm filmler için çağrılması en kötü durumda O(u\*f)’tir.

Sonrasında en çok kişi tarafından izlenen filmlerden başlanarak filmlerin kullanıcı tarafından izlenip izlenmediği kontrol edilir. Kullanıcının tüm filmleri izlemiş olduğu en kötü durumda buradaki karmaşıklık O(f\*f)’tir zira her film izlenmiş olan her filmle tek tek karşılaştırılacaktır.

Bu üç karmaşıklığı birlikte değerlendirdiğimizde fonksiyonun karmaşıklığı en kötü durumda O[(maks(u,f))^2)]dir. Bunun yanında quick sort’un en kötü durumu vermesi de elimizdeki veriler göz önünde bulundurulduğunda bir kullanıcının tüm filmleri izlemiş olması da düşük ihtimallerdir.

Ortalama Çalışma Süresi: 0.002 saniye

**SimilarUserReccom:** Pek çok fonksiyon çağırdığı için önce bu çağrılan fonksiyonları incelemek daha sağlıklı olacaktır.

AddSimNode, searchSimNode ve delSimNode fonksiyonları işlemleri bir bağlı liste için yaparlar, dolayısıyla karmaşıklıkları O(n)’dir.

SimilarUser: Tüm kullanıcıların tüm filmleri izlediği en kötü durumda karmaşıklığı O(u\*f)tir.

mostSimilar: İçerdiği SimilarUser fonksiyonundan daha karmaşık bir ifade içermemektedir.

Bu doğrultuda SimilarUserReccom fonksiyonunun karmaşıklığı iki kullanıcının da tüm filmleri izlemiş olduğu en kötü durumda O(maks(f\*f, u\*f))tir. f\*f’in sebebi bir userın izlemiş olduğu filmleri diğer user’ın izleyip izlememiş olduğunun kontrol edilmesidir. O(u\*f) ise çağrılan MostSimilar fonksiyonundan kaynaklanır.

Ortalama Çalışma Süresi: 0.012 saniye

**ShortestWeighPathRec:** İçerdiği dijkstra algoritmasının basit bir priority queue yapısı kullanmasından dolayı karmaşıklığı O((u+f)^2)’dir.

Ortalama Çalışma Süresi: 0.050 saniye

**CloseAverageRec:**

popularWatchedFilms fonksiyonu watchedFilms fonksiyonunu, kullanıcı tarafından izlenmiş her bir film için watchedBy ve quickSort fonskiyonlarını çağırır dolayısıyla karmaşıklığı O(maks(u\*f, u\*u)’dur.

UsersWatchedThoseFilms fonksiyonu popularWatchedFilms fonksiyonunu çağırır. Bunun yanında her kullanıcı için belli filmleri aradığı bir döngüsü vardır, bunun karmaşıklığı da O(u\*f)dir. Dolayısıyla genel karmaşıklık yine O(maks(u\*f, u\*u)dur.

ClosestAverage fonksiyonu UsersWatchedThoseFilms’i çağırır. Daha karmaşık bir işlem içermemektedir.

CloseAverageRec fonksiyonu ClosestAverage’ı çağırdığı ve diğer tüm öneri fonksiyonları gibi filmlerin kullanıcılar tarafından izlenip izlenmediğini kontrol ettiği için karmaşıklığı O(maks(f\*f, f\*u)dur.

Ortalama Çalışma Süresi: 0.003 saniye

**bestRatingRec:**

PointCalcul aldığı filmi izleyen tüm kullanıcıları gezdiği için karmaşıklığı O(u)dur.

insertionSort’un karmaşıklığı O(f\*f)tir.

MovieList; PointCalcul ve insertionSort’u çağırır.

bestRatingRec, MovieList’i çağırdığından ve filmler kullanıcı tarafından izlendi mi diye baktığından dolayı karmaşıklığı en kötü durumda O(f\*f)’tir.

Ortalama Çalışma Süresi: 0.007 saniye

Not: Ortalama çalışma süreleri fonksiyon başına 10 film önerisi yapılarak ölçülmüştür ve tek bir graf kullanılmıştır. Rapordaki çizimler fare ile yapılmıştır.