**T.C**  
  
**ESKİŞEHİR OSMANGAZİ ÜNİVERSİTESİ**  
  
**MÜHENDİSLİK MİMARLIK FAKÜLTESİ**  
  
  
  
Çizge Kuramı Dersi  
2024-25 Güz Yarıyılı   
Proje Final Raporu  
  
  
  
  
  
  
**Floyd - Warshall**  
**Algoritması**  
  
  
  
  
Baturhan Çağatay 152120211060   
Emre Güner 152120211090

İÇİNDEKİLER

[1. Giriş: 3](#_Toc1325285886)

[2. Algoritma Tarihi: 4](#_Toc598630104)

[3. Algoritmanın Matematiksel Temelleri 5](#_Toc131574247)

[Gereksiz Hesaplamaların Azaltılması 5](#_Toc612151837)

[Dikdörtgen Tanımlarla Optimizasyon 6](#_Toc1719796817)

[Güncellemelerin Minimizasyonu 6](#_Toc282355692)

[4. Algoritmanın Avantajları ve Dezavantajları 6](#_Toc1036351002)

[Avantajları: 6](#_Toc926139019)

[Dezavantajları: 7](#_Toc931146714)

[5. Modern Teknolojilerde Floyd-Warshall Algoritmasının Yeri ve Uygulamaları 7](#_Toc569568670)

[6. Algoritmanın Uygulama Alanları 8](#_Toc1609801506)

[7. Algoritmanın Yapılış Aşamaları 9](#_Toc728035367)

[8. Örnek Uygulama 11](#_Toc1549025440)

[Kaynakça: 16](#_Toc345517498)

Özet

Floyd-Warshall algoritması, graflardaki tüm düğüm çiftleri arasındaki en kısa yolları bulmak için dinamik programlama yöntemini kullanan etkili bir algoritmadır. Algoritma, başlangıçta düğümler arasındaki mesafeleri temsil eden bir mesafe matrisi (D) oluşturur. Her iterasyonda bir düğüm ara düğüm (k) olarak seçilir ve diğer düğüm çiftleri arasındaki mesafelerin bu düğüm üzerinden daha kısa olup olmadığı kontrol edilir. Bu işlem şu aşamalarla gerçekleştirilir:

Düğüm çiftleri arasındaki mevcut mesafeleri içeren bir matris hazırlanır. Doğrudan bağlantı olmayan düğümler arasındaki mesafeler "sonsuz" olarak kabul edilir. Bir düğümün kendisine olan mesafesi sıfırdır.Algoritma, her iterasyonda bir düğümü ara düğüm olarak kullanır ve diğer düğüm çiftleri arasındaki mesafeleri günceller. Güncelleme işlemi şu formülle yapılır:

**D[i][j] = min(D[i][j], D[i][k] + D[k][j])**

Bu formül, i düğümünden j düğümüne doğrudan mesafeyle, k düğümü üzerinden gidilen mesafeyi karşılaştırarak daha kısa olanı seçer. Eğer bir düğümün kendisine olan mesafesi negatif bir değere sahipse (D[i][i] < 0), bu durum negatif döngünün varlığını gösterir. Eğer negatif bir döngü varsa algoritma doğru çıktı veremez. Tüm iterasyonlar tamamlandığında, matrisin her bir hücresi, ilgili düğüm çiftleri arasındaki en kısa mesafeyi içerir.

Algoritmanın avantajları arasında negatif ağırlıklı kenarları desteklemesi ve tüm düğüm çiftleri için en kısa yolları tek bir çalıştırmada bulması bulunur. Dezavantajları ise O(V³) zaman karmaşıklığı ve büyük graflarda yüksek bellek ihtiyacıdır. Modern GPU hızlandırma ve paralel işlem teknikleri, algoritmanın performansını artırmıştır.

Floyd-Warshall algoritması, ulaşım planlaması, sosyal ağ analizi, finansal risk yönetimi, biyoinformatik ve robotik gibi geniş bir uygulama yelpazesinde kullanılmaktadır. Bu algoritma, sadece teorik bir araç değil, aynı zamanda pratik çözümler sunan güçlü bir yöntemdir.

|  |
| --- |
| Sık Kullanılan Kelimeler |
| Algoritma |
| Floyd-Warshall |
| Mesafe |
| Düğüm |
| Kısa Yol |
| Matriks |
| Graf |
| Optimizasyon |

## **1. Giriş:**

En kısa yol problemi, bir başlangıç düğümünden bir hedef düğüme kadar olan en kısa yolu veya rotayı bulma problemidir [1]. Bu problem günlük hayatta karşımıza

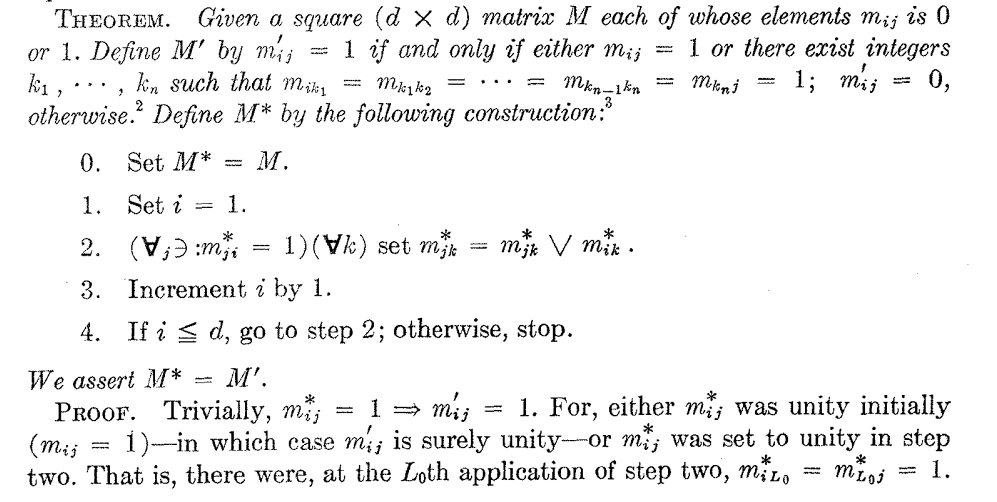
* Ulaşım ve yol planlama,
* Ağ optimizasyonu,
* Enerji sistemleri,
* Sosyal ağ analizleri,
* Oyun geliştirme,
* Kaynak yönetimi,
* Biyoenformatik,
* Sağlık gibi alanlarda çıkar.

Geniş bir uygulama yelpazesi sayesinde, en kısa yol problemi hem teorik hem de pratik açıdan önemli bir araştırma alanı olarak kabul edilmektedir.

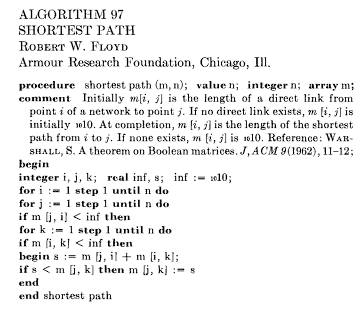
Bu problemi çözmek için zaman içinde birçok algoritma geliştirilmiştir. Bu algoritmalardan biri olan Floyd-Warshall Algoritması tüm düğüm çiftleri arasındaki tüm en kısa yolları tek çalıştırmada bulan ve negatif ağırlıklı kenar varlığını destekleyen bir algoritmadır. Bu algoritma her düğüm çiftleri için en kısa yolu hesaplar ve kenarlar arasında negatif ağırlıklı kenarlar bulunursa bile doğru sonuçlar verebilir. Ancak, bu algoritma negatif döngülerin bulunduğu graflarda doğru sonuç veremez, çünkü negatif döngüler sürekli olarak daha kısa yollar üretmeye yol açar.

## **2. Algoritma Tarihi:**

Algoritmanın geliştirilmesinde Robert Floyd ve Stephen Warshall’ın katkıları büyüktür.

Stephen Warshall 1962 yılında “A THEOREM ON BOOLEAN MATRICES” makalesinde “**yönlendirilmiş graflarda erişilebilirlik matrislerini hesaplamayı”** açıkladı**.** Bu algoritmada Warshall, doğrusal cebir ve matris güncellemeleri kullanılarak erişilebilirlik problemini çözmekte önemli bir adım atmıştır. Ancak, bu algoritma yalnızca **erişilebilirlik** problemini ele alıyordu ve mesafeleri hesaba katmıyordu.  Şekil-1: A Theroem On Boolean Matrices makalesindeki teorem. [3]

Robert Floyd ise 1962 yılında “ALGORITHM 97: SHORTEST PATH” makalesinde dinamik programlama ile tüm düğüm çiftleri arasındaki **en kısa yolları** bulmaya yönelik bir algoritma geliştirdi. Floyd’un algoritması, Warshall’ın erişilebilirlik algoritmasına benzer bir şekilde, matris tabanlı bir güncelleme yöntemi kullandı. Ancak Floyd, **ağırlıklı graflar** için çözüm sağlayarak algoritmayı bir adım öteye taşıdı. Bu algoritma, negatif ağırlıklı kenarların bulunduğu graflarda bile doğru sonuç verebiliyordu (ancak negatif döngülerin bulunduğu durumları ele alamıyordu). Bu özellik, algoritmanın daha geniş bir yelpazede kullanılmasını sağladı.

  
 Şekil-2: ALGORITHM 97: SHORTEST PATH [4]

## **3. Algoritmanın Matematiksel Temelleri**

Floyd-Warshall algoritması, graflardaki tüm çiftler arasındaki en kısa yolları bulmak için kullanılan bir dinamik programlama yöntemidir. Bu algoritma, aşağıdaki tanımlarla başlar:

* **D Matrisleri (Mesafe Matrisi):** Her iki düğüm arasındaki mesafeleri tutan bir kare matris.
* **S Matrisleri (Yol Bilgi Matrisi):** En kısa yol üzerindeki ara düğüm bilgilerini saklayan bir kare matris.

Algoritma iteratif bir yapıyla çalışır ve her iterasyonda (**k** sayıcısıyla belirtilir) D ve S matrisleri güncellenir. k sayıcı 0 ile başlayıp toplam düğüm sayısı olan n'e kadar iterasyon yapar.

### **Gereksiz Hesaplamaların Azaltılması**

Eğer D matrisinde herhangi bir satır veya sütun tamamen "sonsuz" (∞) değerler içeriyorsa, bu satır veya sütundaki diğer değerler daha fazla güncellenmeyebilir.

* Bu durum, bir bağlantının bulunmadığını ve mevcut yol bilgisine yeni bir bilgi eklenemeyeceğini gösterir.
* Söz konusu satır ve sütun, bir önceki iterasyonda (k-1. iterasyon) elde edilen değerlerle sabitlenir.
* Bu yöntem, gereksiz hesaplamaların önünü keserek performansı artırır.

### **Dikdörtgen Tanımlarla Optimizasyon**

Eğer hala tabloda eksik değerler varsa matrixler kullanılarak hesaplanır:

* Diyagonal matris elemanları (**Di**) sıfır olarak kabul edilir ve dikdörtgenin sol üst köşesini oluşturur.
* Dikdörtgen, matrisin k. satırı ve k. sütunu boyunca uzanarak kritik bağlantıları hesaplar.
* Ancak, herhangi bir iterasyonda hem k. satır hem de k. sütun üzerinde aynı anda işlem yapılmaz; bu durum bir sonraki iterasyona bırakılır.

Bu prosedür, hesaplamalarda çifteleme riskini azaltır ve algoritmanın kritik bölgelerine odaklanarak performansı optimize eder.

### **Güncellemelerin Minimizasyonu**

Eğer D matrisindeki bir eleman önceki iterasyonda değişmemişse, ilgili elemanın yer aldığı S matrisi de güncellenmez. Bu bağlantı:

* Hesaplama maliyetini düşürerek sıklıkla tekrarlanan işlemleri önler.
* Eğer bir değer değişirse, ilgili S matrisi güncel k değeriyle tekrar hesaplanır.

Bu prosedür, D ve S matrisleri arasındaki matematiksel bağlılığı netleştirirken performans artışı sağlar. [2]

## **4. Algoritmanın Avantajları ve Dezavantajları**

### 

### Avantajları:

1. **Gereksiz Hesaplamaların Azaltılması:** Hızlandırıcı prosedürler, hesaplama maliyetini düşürerek daha verimli bir performans sağlar.
2. **Dikdörtgen Optimizasyonu:** Algoritmanın dikkati, matrisin kritik bölgelerine yoğunlaştırılır, bu da özellikle büyük boyutlu veri setlerinde hız artışı sağlar.
3. **Optimize Edilmiş Performans:** Sonsuz (∞) değerlerin sıklıkla bulunduğu matrislerde algılanan verim artışı, algoritmanın uygulama alanlarını genişleterek pratik kullanımını arttırır.
4. **Tüm Çiftler Üzerinde**: Tek çalıştırma ile Tüm çiftler arasındaki en kısa yolları bulur. Bu diğer algoritmalardan önemli farkıdır
5. **Dinamik programlama prensibi:** Floyd-Warshall algoritması, dinamik programlama prensibini kullanır. Bu prensip, büyük bir problemin alt problemlere bölünmesini ve bu alt problemlerin çözülerek ana problemin çözümüne ulaşılmasını sağlar. Ayrıca, algoritmanın matris tabanlı bir yapıya sahip olması, uygulanmasını ve anlaşılmasını kolaylaştırır.
6. **Negatif ağırlıklı kenarlara izin verir**: Negatif ağırlıklı kenarları işleyebilir (negatif döngüler hariç). Bu, diğer birçok algoritmanın işleyemediği durumlarda Floyd-Warshall’ı öne çıkarır.
7. **Hem yönlü hem yönsüz graflar:** Floyd-Warshall algoritması, yönlendirilmiş (directed) veya yönlendirilmemiş (undirected) grafiklerde uygulanabilir. Bu, algoritmayı çeşitli problem türlerinde esnek ve kullanılabilir hale getirir.
8. **Ara düğümleri belirler ve yol bilgisi sağlar:** Algoritma sadece mesafeleri değil, aynı zamanda bir düğümden diğerine giderken izlenmesi gereken ara düğümleri de belirler. Bu, sadece en kısa mesafeyi bulmakla kalmayıp, aynı zamanda rotanın tamamını analiz etmek isteyen uygulamalar için önemli bir özelliktir.

### Dezavantajları:

1. **Zaman Karmaşıklığı (O(V³))**: Floyd-Warshall algoritmasının zaman karmaşıklığı **(O(V³))** olduğundan, büyük ölçekli grafiklerde performansı düşer. Bu, algoritmayı daha büyük grafiklerde yavaş hale getirir.
2. **Negatif Döngülerle Sınırlı**: Algoritma negatif döngüleri tespit edebilse de, bu tür grafiklerde doğru bir çözüm sunamaz. Negatif döngü varlığı durumunda sonuçlar anlamsız olur.
3. **Hafıza Gereksinimi**: Algoritma, düğümler arası mesafeleri saklamak için bir V x V matris kullanır. Bu durum, büyük grafiklerde yüksek hafıza tüketimine neden olabilir.
4. **Dinamik Güncellemeler İçin Uygun Değil**: Algoritma, statik grafiklerde daha etkilidir. Grafikteki bir değişiklik (yeni bir kenar eklenmesi veya kaldırılması) durumunda tüm işlemin tekrar edilmesi gerekir.
5. **Alternatiflerin Daha Hızlı Olduğu Durumlar**: Tek kaynaklı en kısa yol problemleri için (örneğin, belirli bir düğümden diğer tüm düğümlere en kısa yolu bulma) Dijkstra algoritması gibi yöntemler genellikle daha hızlıdır.

## **5. Modern Teknolojilerde Floyd-Warshall Algoritmasının Yeri ve Uygulamaları**

Floyd-Warshall algoritması, grafik teorisi ve algoritmalar alanında, tüm düğüm çiftleri arasındaki en kısa yolları hesaplama yeteneği ile öne çıkmaktadır. Bu özellik, algoritmanın modern teknolojilerde geniş bir kullanım alanına sahip olmasını sağlamaktadır. Özellikle, büyük veri setlerinin analiz edilmesi ve karmaşık ağ yapılarına çözüm bulunması gereken alanlarda, algoritma etkili bir araç olarak kullanılmaktadır. Son yıllarda, donanım hızlandırma tekniklerindeki gelişmeler, Floyd-Warshall algoritmasının performansını daha da artırmıştır. Grafik işlemciler (GPU'lar) ve paralel işlem mimarileri bu bağlamda öne çıkan unsurlar arasında yer almaktadır.

**Donanım Hızlandırma ve GPU Uygulamaları:**

Grafik işlemcilerin (GPU'lar) paralel işlem yetenekleri, Floyd-Warshall algoritmasının büyük ölçekli veri setlerinde daha verimli çalışmasını sağlamaktadır. GPU'lar, aynı anda birden fazla işlem gerçekleştirebilme yeteneği sayesinde, algoritmanın paralel bir şekilde uygulanmasını mümkün kılmaktadır. Örneğin, Anjary'nin çalışmasında, algoritma 3D tensörler ve GPU'lar kullanılarak yeniden uygulanmış ve bu sayede hesaplama süresinde belirgin bir iyileşme sağlanmıştır. [6] Benzer şekilde, Lund ve Smith'in araştırmasında, CUDA platformu üzerinde çok aşamalı bir çekirdek kullanılarak Floyd-Warshall algoritması uygulanmış ve önceki yöntemlere göre yaklaşık 5 kat hız artışı elde edilmiştir. [7] Ayrıca, Oleg Konings tarafından geliştirilen basit bir CUDA tabanlı Floyd-Warshall algoritması uygulaması, GPU'ların paralel işlem yeteneklerinden faydalanarak performans iyileştirmeleri sağlamaktadır. [8]

**Paralel Hesaplama ve Çok Çekirdekli İşlemciler:**

Paralel hesaplama, modern işlemci mimarilerinin sağladığı çoklu çekirdek yapılarıyla Floyd-Warshall algoritmasının daha verimli bir şekilde çalışmasını mümkün kılmaktadır. Çok çekirdekli işlemciler üzerinde yapılan optimizasyonlar, algoritmanın geniş çaplı ağlar üzerinde hızlı sonuçlar üretmesini sağlamaktadır.

Floyd-Warshall algoritmasının paralel hesaplama yetenekleri, çok çekirdekli işlemciler üzerinde de optimize edilmektedir. Sebastián Calderón, Emilio Rucci ve Federico Chichizola’nın çalışmasında, Floyd-Warshall algoritması x86 mimarisi üzerinde OpenMP teknolojisi kullanılarak optimize edilmiştir. Bu optimizasyon, algoritmanın tüm çiftler için en kısa yol hesaplamalarını hızlandırmış ve geleneksel CPU tabanlı uygulamalara kıyasla önemli performans iyileştirmeleri elde edilmiştir. Çalışma, algoritmanın çok çekirdekli işlemcilerdeki etkili kullanımına dair bir örnek oluşturmaktadır. [9]

## **6. Algoritmanın Uygulama Alanları**

* **Şehir İçi Yol Ağı ve Taşımacılık Planlaması:**

Floyd-Warshall algoritması, şehir içi ulaşım ağlarının optimizasyonunda etkin bir şekilde kullanılmaktadır. Keskin ve Özcan'ın çalışmalarında, Türkiye'deki lojistik merkezler arasında en kısa yolların belirlenmesi amacıyla algoritma kullanılmış ve bu sayede taşımacılık planlamasında verimlilik artırılmıştır. [7]

* **Sosyal Ağ Analizi:**

Sosyal ağlarda, kullanıcılar arasındaki etkileşimlerin analizi önem taşır. Floyd-Warshall algoritması, ağdaki tüm kullanıcı çiftleri arasındaki en kısa etkileşim yollarını hesaplayarak, ağın yapısal özelliklerinin anlaşılmasına yardımcı olur. Bu sayede, bilgi yayılımı, etki analizi ve topluluk tespiti gibi konularda derinlemesine analizler yapılabilir.

* **Finansal Ağlar ve Risk Analizi:**

Finansal sistemlerde, kurumlar arasındaki ilişkiler ve para akışları karmaşık ağlar oluşturur. Floyd-Warshall algoritması, bu ağlarda tüm kurum çiftleri arasındaki en kısa finansal bağlantıları belirleyerek, sistemik risklerin tespiti ve finansal istikrarın değerlendirilmesinde kullanılabilir. Örneğin, bir çalışmada, risk arbitrajı için modifiye edilmiş Floyd-Warshall algoritması kullanılarak finansal risklerin analizi yapılmıştır. [10]

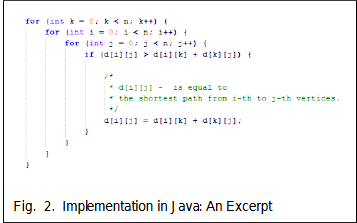
* **Biyoinformatik ve Genomik Araştırmalar:**

Biyoinformatik alanında, genler ve proteinler arasındaki etkileşimler ağ yapılarıyla modellenir. Floyd-Warshall algoritması, bu biyolojik ağlarda tüm molekül çiftleri arasındaki en kısa etkileşim yollarını hesaplayarak, hücresel süreçlerin anlaşılmasına katkı sağlar. Bu sayede, hastalıkların moleküler mekanizmaları ve potansiyel tedavi hedefleri belirlenebilir.

* **Robotik Yol Planlaması ve Otonom Sistemler:**

Otonom robotlar ve araçlar, çevrelerinde güvenli ve verimli bir şekilde hareket edebilmek için yol planlaması yapmalıdır. Floyd-Warshall algoritması, robotların bulundukları ortamdaki tüm konum çiftleri arasındaki en kısa yolları hesaplayarak, engellerden kaçınma ve hedefe ulaşma süreçlerinde kullanılır. Bu sayede, otonom sistemlerin navigasyon kabiliyetleri geliştirilir.

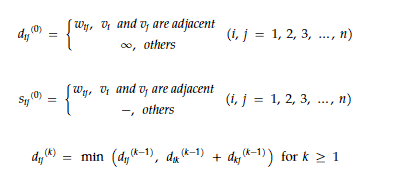
## **7. Algoritmanın Yapılış Aşamaları**



Şekil-3: A Review And Evaluations Of Shortest Path Algorithms [11]

Algoritma, üç düğümü ele alarak çalışır:

* i: Başlangıç düğümü
* j: Hedef düğümü
* k: Ara düğüm



Şekil-4: Modelling the Shortest Path for Inner Warehouse Travelling Using the Floyd–Warshall Algorithm [2]

Algoritmanın yapılış aşamalarının detaylı gösterimi:

1. Başlangıç Matrisinin Oluşturulması

İlk adımda, grafik düğümleri arasındaki mesafeleri temsil eden bir matris D oluşturulur. Bu matrisin başlangıç değerleri aşağıdaki kurallara göre atanır:

Eğer iki düğüm arasında bir kenar varsa, matrisin ilgili hücresine bu kenarın ağırlığı atanır. Örneğin, D[i][j] hücresine, i ve j düğümleri arasındaki kenarın ağırlığı yazılır.

Eğer iki düğüm arasında bir kenar yoksa, bu düğümler arasındaki mesafe sonsuz olarak kabul edilir.

Bir düğümün kendisine olan mesafesi sıfır olarak atanır. Örneğin, D[i][i] = 0.

Bu matris, başlangıçta grafın doğrudan bağlantılarını ve mesafelerini temsil eder. Bu yapı, algoritmanın diğer adımlarında temel alınır.

2. Döngülerle Matrisin Güncellenmesi

Matrisin güncellenmesi için, üç seviyeli iç içe geçmiş döngüler kullanılır. Bu döngüler, her bir düğüm üzerinden geçerek en kısa mesafeleri günceller:

Dış Döngü:  
 Bu döngü, algoritmanın her iterasyonunda bir ara düğüm k belirler. k düğümü, diğer tüm düğümler arasındaki yolların olası bir ara düğüm olarak kullanılmasını sağlar.

İlk İç Döngü:  
 İlk iç döngü, başlangıç düğümünü i seçer. Bu döngü, k ara düğümünü kullanarak i düğümünden diğer düğümlere olan mesafelerin kontrol edilmesini sağlar.

İkinci İç Döngü:  
 İkinci iç döngü, hedef düğümü j seçer. Bu döngü, i düğümünden j düğümüne olan mevcut mesafenin, k ara düğümünü kullanarak daha kısa bir mesafeyle değiştirilip değiştirilemeyeceğini kontrol eder.

Bu adımda, her bir hücrede D[i][j] = min(D[i][j], D[i][k] + D[k][j]) formülü kullanılarak mevcut mesafeler güncellenir. Bu formül, doğrudan i ile j arasında bir yol olup olmadığını ve k ara düğümünün bu mesafeyi kısaltıp kısaltamayacağını kontrol eder.

3. Daha Kısa Bir Yol Bulunduğunda Güncelleme

İç içe döngülerin her bir turunda, daha kısa bir yol bulunması durumunda matris güncellenir:

Eğer D[i][j] > D[i][k] + D[k][j] ise, D[i][j] değeri D[i][k] + D[k][j] ile değiştirilir.

Bu işlem, düğümler arasındaki mevcut en kısa mesafenin sürekli iyileştirilmesini sağlar.

Bu güncelleme işlemi, her iterasyonda matrisin daha doğru ve optimize edilmiş bir hal almasını sağlar.

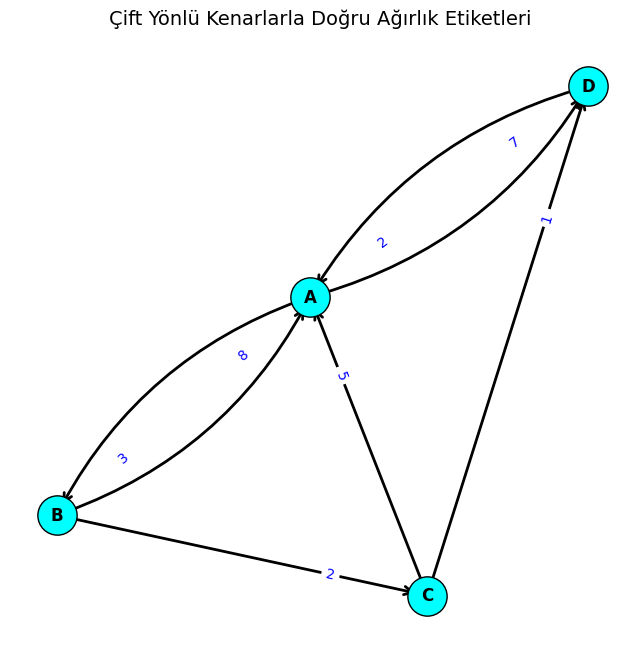
4. Tüm Döngülerin Tamamlanması ve Nihai Sonuç

Son adımda, tüm düğüm çiftleri üzerinden yapılan güncellemeler tamamlanır. Üç seviyeli iç içe döngülerin tamamlanmasının ardından:

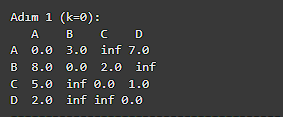
D matrisindeki her bir hücre D[i][j], i ve j düğümleri arasındaki en kısa mesafeyi içerir. Eğer D[i][j] sonsuz ise, bu durum i ve j düğümleri arasında bir yol bulunmadığını gösterir. Sonuç olarak, algoritma, düğüm çiftleri arasındaki tüm en kısa yolların hesaplanmasını sağlayan nihai bir mesafe matrisi üretir.

## **8. Örnek Uygulama**

Bu bölümde, Floyd-Warshall algoritmasının çalışması, Şekil-5’teki graf üzerinde adım adım incelenmiştir. Aşağıdaki şekiller ve matrisler, algoritmanın her iterasyonunda meydana gelen güncellemeleri göstermektedir.

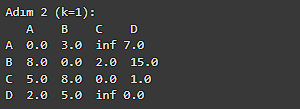
 

Şekil-5: Uygulamamızda kullanılan graf Şekil-6: Kenarlar tablosu



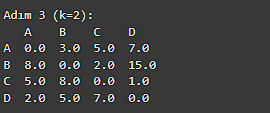
Şekil-7: Oluşan kenarlar tablosuna göre mesafe matriksi

İlk aşamada (k=0), Şekil-7’deki başlangıç mesafe matrisi oluşturulmuştur. Bu matris, kenar tablosunda belirtilen mesafelere dayanır. İki düğüm arasında bir kenar varsa bu kenarın ağırlığı matrisin ilgili hücresine atanmıştır. Eğer iki düğüm arasında bir kenar yoksa, mesafe "sonsuz" (inf) olarak tanımlanmıştır. Aynı düğümün kendisine olan mesafesi ise sıfır olarak atanmıştır. Bu adımda, henüz ara düğümler dikkate alınmadığı için sadece doğrudan bağlantılar üzerinden mesafeler belirtilmiştir.



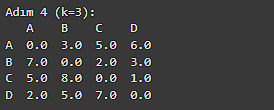
Şekil-8

İkinci aşamada (k=1), A düğümü ara düğüm olarak kullanılarak matris güncellenmiştir. Bu iterasyonda, diğer düğümler arasındaki mesafeler A düğümü üzerinden kontrol edilmiştir. Eğer bir düğüm çifti arasında A düğümü aracılığıyla daha kısa bir yol bulunuyorsa, mesafe matrisindeki değer buna göre güncellenmiştir. Güncellenen tüm değerler Şekil-8 tablosunda gösterilmektedir. Örneğin, B düğümünden C düğümüne olan mesafe incelendiğinde, bu mesafenin direkt mi ((B, C) = 2) yoksa ara düğüm olarak A üzerinden mi daha kısa olduğu kontrol edilmiştir. A üzerinden hesaplama yapılırken, B’den A’ya olan mesafe ((B, A )= 8) ile A’dan C’ye olan mesafe ((A, C) = inf) toplanmıştır ve toplam değer sonsuz (inf) olmuştur. Bu durumda, B ve C düğümleri arasındaki en kısa mesafe 2 olarak kalmıştır. Böylece, matris hücresindeki B düğümünden C düğümüne olan mesafesi 2 olarak korunmuştur.



Şekil-9

Üçüncü aşamada (k=2), B düğümü ara düğüm olarak kullanılmıştır. B düğümünden geçen daha kısa yollar, mesafe matrisindeki ilgili hücrelerde güncellenmiştir. Eğer B düğümünü ara düğüm olarak kullanarak yeni bir yol keşfedilmişse, bu yol matrisin ilgili hücresinde belirtilmiştir. Örneğin A düğümü ile C düğümü arasındaki en kısa yol incelenmesinde A’dan C’ye direkt yol olmadığı için ‘inf’ değeri iken, B düğümünü ara düğüm olarak kullanıp A düğümünden B düğümüne ardından da B düğümünden C düğümüne giden yolun hesaplanması yapılınca 3+2 işlemi sonucu olan 5 daha küçük değer olduğu için (A, C) arası 5 olarak güncellenir. Oluşan tüm değerler Şekil-9 tablosunda gözlemlenir.



Şekil-10

Dördüncü aşamada (k=3), C düğümü ara düğüm olarak kullanılarak matris yeniden Şekil-10’daki gibi güncellenmiştir. C düğümü, diğer düğümler arasında daha kısa yollar sağlayabilmektedir. Eğer bu iterasyonda herhangi bir hücrede daha kısa bir yol bulunmuşsa, bu değer matrisin ilgili hücresinde güncellenmiştir. Örneğin, A düğümünden D düğümüne olan mesafe başlangıçta 7 iken C düğümü üzerinden daha kısa bir yol bulunup bulunamayacağı kontrol edilir. Bu işlem için (A, C) = 5, (C, D) = 1 olduğu için 6<7 kuralı sağlandığından (A, D) = 6 olarak güncellenmiştir.



Şekil-11

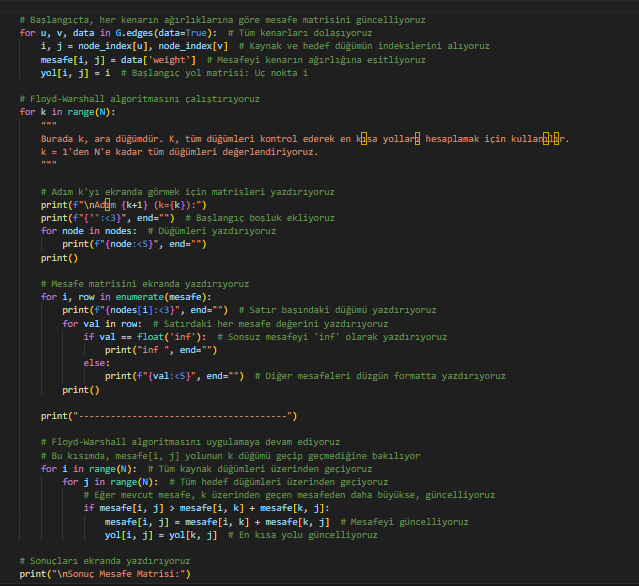
Tüm iterasyonlar tamamlandığında, Şekil-11’deki sonuç mesafe matrisi elde edilmiştir. Bu matris, grafın tüm düğüm çiftleri arasındaki en kısa yolları içermektedir. Matrisin her bir hücresi, ilgili düğüm çiftleri arasındaki en kısa mesafeyi temsil etmektedir.

Sonuç olarak:

* A’dan B’ye en kısa mesafe 3 birimdir.
* A’dan C’ye en kısa mesafe 5 birimdir.
* A’dan D’ye en kısa mesafe 6 birimdir.
* B’den A’ya en kısa mesafe 5 birimdir.
* B’den C’ye en kısa mesafe 2 birimdir.
* B’den D’ye en kısa mesafe 3 birimdir.
* C’den A’ya en kısa mesafe 3 birimdir.
* C’den B’ye en kısa mesafe 6 birimdir.
* C’den D’ye en kısa mesafe 1 birimdir.
* D’den A’ya en kısa mesafe 2 birimdir.
* D’den B’ye en kısa mesafe 5 birimdir.
* D’den C’ye en kısa mesafe 7 birimdir.

! B düğümünden D düğümüne en kısa mesafe 2 birimken D düğümünden B düğümüne en kısa mesafe 5 birimdir. Bu farkın sebebi kullanılan grafın yönlü graf olmasıdır.

! Eğer içinde negatif döngü yapmayacak bir biçimde negatif kenar bulundurursa yapılan işlemlerde herhangi bir değişim olmayacaktır. Ancak negatif döngü aldığında sonuç negatif ve çoğu zaman daha küçük olacağı için algoritmanın verimli ve doğru çalışmasını engeller.

  
Şekil-12: Algoritmanın temel kullanım kodu

## Kaynakça:

[1] A Review and Evaluations of Shortest Path Algorithms - <https://www.researchgate.net/profile/Magzhan-Kairanbay/publication/310594546_A_Review_and_Evaluations_of_Shortest_Path_Algorithms/links/5ac49631a6fdcc1a5bd06106/A-Review-and-Evaluations-of-Shortest-Path-Algorithms.pdf>

[2]- Modelling the Shortest Path for Inner Warehouse Travelling Using the Floyd–Warshall Algorithm -- <https://www.mdpi.com/2227-7390/12/17/2698>

[3]- A Theorem on Boolean Matrices -- <https://dl.acm.org/doi/pdf/10.1145/321105.321107>

[4]- ALGORITHM 97 SHORTEST PATH -- <https://dl.acm.org/doi/pdf/10.1145/367766.368168>

[5] Anjary, T. (2023). *The Floyd-Warshall Algorithm Re-implemented Using 3D-Tensors and Hardware Acceleration*. -- <https://arxiv.org/abs/2310.03983>

[6] Lund, B., & Smith, J. W. (2010). *A Multi-Stage CUDA Kernel for Floyd-Warshall*. -- <https://arxiv.org/abs/1001.4108>

[7] Konings, O. *CUDA implementation of the Floyd-Warshall All pairs shortest path algorithm*. -- <https://github.com/OlegKonings/CUDA_Floyd_Warshall_>

[8] Calderón, S., Rucci, E., & Chichizola, F. (2024). *Enhanced OpenMP Algorithm to Compute All-Pairs Shortest Path on x86 Architectures*. -- <https://arxiv.org/abs/2403.18619>

[9] Keskin, B., & Özcan, E. (2023). *En Kısa Yol Optimizasyonlarında Floyd-Warshall Algoritması: Lojistik Merkezler Örneği*. -- <https://dergipark.org.tr/tr/pub/demiryolu/issue/72163/1187884>

[10] *Modified Floyd-Warshall Algorithm for Risk Arbitrage*. -- <https://wseas.us/e-library/conferences/2010/Corfu/SYSTEMS/SYSTEMS1-47.pdf>

[11] A Review And Evaluations Of Shortest Path Algorithms https://www.researchgate.net/publication/310594546\_A\_Review\_and\_Evaluations\_of\_Shortest\_Path\_Algorithms