# GRAF Algoritmaları ve Uygulamaları

https://github.com/arzuv-hudaynazarova/Graph_Algorithms_Distribution_Route

https://github.com/Joseph0grcn/Graph_Algorithms_Distribution_Route

**Ad:** Arzuv
**Soyadı:** Hudaynazarova
**Öğrenci-No:** 5200505059

**Ad:** Yusuf
**Soyadı:** Gürcan
**Öğrenci-No:** 1180505810

## Graf Teorisi ve Algoritmaları:

Graf teorisi, matematik ve bilgisayar bilimlerinde yaygın olarak kullanılan bir alan olup, nesneler arasındaki ilişkileri modellemeye yarayan bir araçtır. Graf teorisi, matematiksel yapıları ve bu yapıların özelliklerini inceleyen bir matematik dalıdır. Bir graf, düğüm (nokta) ve kenar (çizgi) adı verilen iki temel bileşenden oluşur. Düğümler, belirli bir alandaki nesneleri (örneğin şehirler, insanlar, bilgisayarlar) temsil ederken, kenarlar bu nesneler arasındaki ilişkileri (örneğin yollar, arkadaşlık bağlantıları, ağ bağlantıları) gösterir. Graf teorisi, bu yapılar üzerinde çalışarak çeşitli problemleri çözmeyi amaçlar.

Graf algoritmaları, graf teorisi alanında kullanılan yöntemlerdir ve temel problemleri çözmeye odaklanır. Bu problemler arasında en kısa yol, en düşük maliyetli ağ, maksimum akış, minimum kesme, renklendirme ve dolaşma gibi konular bulunur. Bu tür algoritmalar, ulaşım, ağ planlaması ve optimizasyon gibi alanlarda önemli uygulamalara sahiptir.

Örneğin, bu projede, belirli bir alandaki farklı noktalar arasındaki en kısa yolları bulmak için bir graf algoritması kullanılacaktır. Bu algoritma, verilen bir noktadan diğerine giderken en kısa ve en hızlı yolun ne olduğunu belirlemeye yardımcı olabilir. Bu tür bir algoritma, sürücülerin en hızlı rota seçeneklerini belirlemesine yardımcı olarak trafik yoğunluğunu azaltmaya ve yakıt tasarrufu sağlamaya katkıda bulunabilir.

**Graf teorisinde kullanılan temel kavramlar şunlardır:**

**1. Düğüm (vertex):** Grafın temel birimi olan düğüm, genellikle bir nokta ile gösterilir ve belirli bir alandaki nesneleri (örneğin şehirler, insanlar, bilgisayarlar) temsil eder.

**2. Kenar (edge):** İki düğüm arasındaki bağlantıyı temsil eden kenar, genellikle bir çizgi ile gösterilir ve bu nesneler arasındaki ilişkileri (örneğin yollar, arkadaşlık bağlantıları, ağ bağlantıları) gösterir.

**3. Ağırlıklı Graf:** Kenarlarına ağırlık değeri atanmış graf türüdür. Ağırlıklar, kenarlar arasındaki ilişkinin önemini, maliyetini veya mesafesini temsil edebilir.

**4. Yönlü Graf (digraph):** Kenarlarının yönü olan graf türüdür. Yönlü graf, düğümler arasındaki ilişkinin tek yönlü olduğu durumları modellemek için kullanılır, örneğin sosyal medya takipçi ilişkileri veya trafik yönleri.

**5. Çevre (cycle):** Başlangıç ve bitiş düğümü aynı olan bir düğüm dizisidir. Çevre, graf içinde belirli bir düğümden başlayarak ve aynı düğümde biterek dolaşan bir yol anlamına gelir.

**6. Altgraf (subgraph):** Bir grafın düğüm ve kenarlarının bir alt kümesinden oluşan graf türüdür. Altgraf, orijinal grafa göre daha küçük ve sadeleştirilmiş bir yapı sunar.

**7. Komşuluk (neighborhood):** Bir düğümün tüm komşu düğümlerinin kümesidir. Bir düğümün komşuları, ona doğrudan bağlı olan diğer düğümlerdir.

**8. Derece (degree):** Bir düğümün bağlı olduğu kenar sayısıdır. Derece, düğümün ne kadar yoğun bağlantılara sahip olduğunu gösterir ve graf analizinde önemli bir rol oynar.

Bu temel kavramlar, graf teorisi ve algoritmalarının temelini oluşturur ve graf yapılarını anlamak ve analiz etmek için kullanılır.

**Aşağıda, bazı önemli graf algoritmaları açıklanmaktadır:**

## 1. Derinlik Öncelikli Arama (DFS, Depth First Search) ve Genişlik Öncelikli Arama (BFS, Breadth First Search):

Bu algoritmalar, en temel graf algoritmaları olup, graf problemlerinin çözümünde sıklıkla kullanılırlar. DFS ve BFS algoritmaları, ağaç yapısı oluşturma, topolojik sıralama, çevre bulma, bağlı bileşenleri tespit etme ve grafın planar olup olmadığını belirleme gibi problemler için kullanılır. Ayrıca, yapay zeka, dil işleme, veri madenciliği, bilgisayar grafikleri, haritalama, yönlendirme, navigasyon, trafik optimizasyonu, oyun yapay zekası, ağ teorisi, biyoinformatik, graf analizi, ağ arama ve sosyal ağ analizi gibi alanlarda da kullanılır.

DFS, ağaç yapısı oluşturmak, topolojik sıralama yapmak, çevre bulmak, bağlı bileşenleri tespit etmek ve grafın planar olup olmadığını belirlemek için kullanılır. Ayrıca, yapay zeka, dil işleme, veri madenciliği ve bilgisayar grafikleri gibi alanlarda da kullanılır.

BFS, genellikle en kısa yol bulma problemlerini çözmek için kullanılır. Haritalama, yönlendirme, navigasyon, trafik optimizasyonu ve oyun yapay zekası gibi birçok uygulama alanında yaygın olarak kullanılır. Ayrıca, ağ teorisi, biyoinformatik, graf analizi ve ağ arama gibi uygulamalarda ve sosyal ağ analizi gibi alanlarda da kullanılır.

### 1.1. Derinlik Öncelikli Arama (DFS, Depth First Search):

DFS, başlangıç düğümünden başlayarak grafi derinlik öncelikli olarak dolaşan bir algoritmadır. Bu algoritma, grafın tüm düğümlerini keşfetmek ve graf yapısını analiz etmek için kullanılır. DFS, bir düğümün tüm dallarını keşfetmeden önce, o düğümle ilişkili en derin düğümlere kadar gitmeyi amaçlar. DFS ayrıca, çevre bulma, bağlı bileşenleri tespit etme ve grafın planar olup olmadığını belirleme gibi problemler için kullanılabilir. Bu algoritma, genellikle rekürsif olarak uygulanır.

### 1.2. Genişlik Öncelikli Arama (BFS, Breadth First Search):

BFS, başlangıç düğümünden başlayarak grafi katmanlar halinde dolaşan bir algoritmadır. BFS, grafın en kısa yolunu bulmak ve grafın bağlantılı bileşenlerini bulmak için kullanılır. BFS, bir düğümün tüm komşularını keşfetmeden önce, aynı derinlikteki tüm düğümleri ziyaret etmeyi amaçlar. BFS, bir kuyruk kullanılarak uygulanır. Bu algoritma, düğümlerin bir önceden belirlenmiş sırayla keşfedilmesi garantisi nedeniyle öngörülebilir bir davranış gösterir.

**2. En Kısa Yol Algoritmaları:** Dijkstra, Bellman-Ford, A* ve Floyd-Warshall gibi en kısa yol algoritmaları sayesinde, haritalama, yönlendirme, navigasyon, trafik optimizasyonu ve daha birçok uygulamada önemli işlemler gerçekleştirilebilir.

### 2.1. Dijkstra Algoritması:

Dijkstra algoritması, ağırlıklı graf üzerinde başlangıç düğümünden diğer tüm düğümlere en kısa yol uzunluklarını hesaplamak için kullanılır. Bu algoritma, grafın kenarlarına pozitif ağırlık değerleri atanmış olması gerekliliği ile birlikte, çevreleri içermemesi durumunda doğru sonuçlar verir. Algoritma, minimum önceliğe sahip bir öncelik kuyruğu veri yapısı kullanır. Başlangıç düğümünün uzaklık değeri 0 olarak ayarlanır ve diğer düğümlerin uzaklık değerleri sonsuz olarak ayarlanır. Dijkstra algoritması, her adımda henüz işlem görmemiş düğümler arasından uzaklık değeri en küçük olan düğümü seçer ve onu işleme alır. Bu düğümün komşularının uzaklık değerleri, başlangıç düğümüne olan uzaklıklarından daha kısa bir yola sahip olup olmadıkları kontrol edilerek güncellenir. Bu işlem, tüm düğümler işlem görene kadar devam eder.

### 2.2. Bellman-Ford Algoritması:

Bellman-Ford algoritması, ağırlıklı graf üzerinde başlangıç düğümünden diğer tüm düğümlere en kısa yol uzunluklarını hesaplamak için kullanılır. Bu algoritma, grafın kenarlarına negatif veya pozitif ağırlık değerleri atanmış olabilir ve ayrıca graf içinde çevrelerin oluşmasına izin verir. Algoritma, dinamik programlama tekniklerine dayanır ve her bir düğüm için en kötü senaryo tahmini yapar. Bellman-Ford algoritması, her adımda tüm kenarları dolaşarak, tüm düğümlerin uzaklık değerlerini günceller. Başlangıç düğümünün uzaklık değeri 0 olarak ayarlanır ve diğer düğümlerin uzaklık değerleri sonsuz olarak ayarlanır. Algoritma, her düğüm için tüm kenarları dolaşarak, diğer düğümlere olan uzaklıklarını günceller. Bu işlem, tüm düğümler için tekrarlanarak en kısa yol uzunlukları bulunur.

### 2.3. A* Algoritması:

A* algoritması, bir graf içindeki iki düğüm arasındaki en kısa yolu bulmak için kullanılan bir arama algoritmasıdır. Bu algoritma, Dijkstra algoritması ile benzerdir, ancak bir heuristik fonksiyonu kullanır. Bu fonksiyon, hedef düğüme olan tahmini uzaklığı hesaplayarak, daha hızlı keşfetmek için kullanılır. A* algoritması, genellikle robotik ve yapay zeka uygulamaları gibi algoritmik problemlerde kullanılır.

### 2.4. Floyd-Warshall Algoritması:

Floyd-Warshall algoritması, bir graf içindeki tüm çiftler arasındaki en kısa yol uzunluğunu hesaplamak için kullanılan dinamik bir programlama algoritmasıdır. Bu algoritma, ağırlıklı ve yönlü bir graf için de kullanılabilir. Floyd-Warshall algoritması, bir matris kullanarak her iki düğüm arasındaki en kısa yol uzunluğunu hesaplar. Algoritma, her bir düğümü ara düğüm olarak kullanır ve tüm ara düğümleri birbirleriyle karşılaştırarak en kısa yolları hesaplar. Bu sayede, tüm çiftler arasındaki en kısa yollar tek seferde hesaplanabilir.

**3. Minimal Kapsayıcı Ağaç (MST) Algoritmaları**: MST algoritmaları, ağırlıklı bir graf üzerinde ağırlıkların toplamı en küçük olan bağlantılı ve çevresiz bir alt grafik bulmayı amaçlar. Bu algoritmalar, telekomünikasyon, yol ağları, elektrik dağıtımı ve mimaride planlama gibi birçok uygulamada önemli rol oynar. Prim ve Kruskal algoritmaları, bu tür problemler için sıklıkla kullanılan yöntemlerdir.

Her iki algoritma da farklı durumlar için avantajlar sunar. Örneğin, yoğun bir graf üzerinde çalışırken Prim algoritması daha hızlı sonuçlar verebilirken, seyrek bir graf üzerinde Kruskal algoritması daha etkili olabilir. MST algoritmalarının seçimi, kullanılacak grafın özelliklerine ve uygulamanın gereksinimlerine bağlıdır.

### 3.1. Prim Algoritması:

Prim algoritması, ağırlıklı bir grafın minimum kapsayıcı ağacını (MST) bulmak için kullanılan bir algoritmadır. Bu algoritma, rastgele seçilen bir başlangıç düğümüyle başlar ve her adımda, mevcut ağaçtan ağaca dahil edilmemiş bir düğüme giden en düşük ağırlıklı kenar seçilir ve ağaca eklenir. Bu işlem, tüm düğümler ağaçta olana kadar devam eder. Prim algoritması, özellikle yoğun grafiklerde etkilidir ve işlem süresi, düğüm sayısıyla orantılıdır.

### 3.2. Kruskal Algoritması:

Kruskal algoritması, ağırlıklı bir grafın minimum kapsayıcı ağacını (MST) bulmak için kullanılan bir algoritmadır. Bu algoritma, kenarları ağırlıklarına göre sıralar ve her adımda en düşük ağırlıklı kenarı ekler, ancak çevre oluşturacak kenarları atlar. Kruskal algoritması, seçkin bir veri yapısı olan birleştirme-bulma kümesi kullanarak çevre oluşumunu kontrol eder. Bu işlem, tüm düğümler ağaçta olana kadar devam eder. Kruskal algoritması, özellikle seyrek grafiklerde etkilidir ve işlem süresi, kenar sayısıyla orantılıdır.

**4. Maksimum Akış Algoritmaları:** Bir ağ üzerindeki kaynak düğümden hedef düğüme doğru en fazla ne kadar akışın sağlanabileceğini belirlemeye yarar. Bu algoritmalar, iletişim ağları, taşımacılık, enerji akışı, ağların optimizasyonu, elektrik ve su yönetimi gibi uygulamalarda yaygın olarak kullanılır.

### 4.1. Ford-Fulkerson Algoritması:

Ford-Fulkerson algoritması, yönlü ve ağırlıklı bir graf üzerinde maksimum akışı bulmak için kullanılır. Bu algoritma, graf üzerindeki kenarların kapasitelerini kullanarak kaynak düğümünden hedef düğüme akan maksimum akışı hesaplar. Algoritma, artırıcı yol bulma ve akışı güncelleme adımlarını tekrarlayarak çalışır. Her adımda, artırıcı bir yol bulunur ve bu yol boyunca akış artırılır. Bu işlem, artırıcı yol kalmayana kadar devam eder. Ford-Fulkerson algoritması, artırıcı yol bulma için farklı yöntemler kullanabilir, ancak algoritmanın zaman karmaşıklığı, kullanılan yönteme bağlıdır.

### 4.2. Edmonds-Karp Algoritması:

Edmonds-Karp algoritması, Ford-Fulkerson algoritmasının bir varyasyonudur ve maksimum akış problemini daha verimli bir şekilde çözmek için geliştirilmiştir. Bu algoritma, artırıcı yolları bulmak için BFS (genişlik öncelikli arama) kullanır. BFS, artırıcı yolun en kısa yol olmasını sağlar, bu da algoritmanın daha verimli çalışmasına yardımcı olur. Edmonds-Karp algoritması, daha iyi bir zaman karmaşıklığı ile daha hızlı sonuçlar üretir. Bu algoritma, daha düşük karmaşıklıkla, yani O(V^3) karmaşıklıkla çalışır, burada V grafın düğüm sayısıdır.

**5. Maksimum Eşleşme Algoritmaları:** İki parçalı graf üzerinde en iyi eşleşmeyi bulmak için kullanılır. Bu algoritmalar, işe alım, evlilik planlaması, görev atamaları gibi alanlarda uygulanabilir. İki önemli maksimum eşleşme algoritması Hopcroft-Karp ve Hungarian algoritmasıdır. Maksimum eşleşme algoritmaları, belirli graf problemlerini çözmek için tasarlanmıştır ve bazı durumlarda, algoritmanın performansını optimize etmek için özel veri yapıları

kullanılır. Örneğin, Hopcroft-Karp algoritması, BFS ve DFS algoritmalarını kullanarak alternatif yolları bulur ve bu yolları tutmak için özel bir veri yapısı kullanır. Hungarian algoritması ise, maliyet matrisini işlemek için etiketleme ve eşleştirme adımlarını kullanır. Bu algoritmaların uygulamaları, ağ analizi, coğrafi bilgi sistemleri, optimizasyon problemleri ve makine öğrenimi gibi geniş bir alan yelpazesinde bulunabilir.

### 5.1. Hopcroft-Karp Algoritması:

Hopcroft-Karp algoritması, iki parçalı grafın maksimum eşleşmesini bulmak için kullanılır. Algoritma, BFS ve DFS (genişlik ve derinlik öncelikli arama) algoritmalarını birleştirerek çalışır. Her adımda, BFS araması kullanılarak, alternatif yollar bulunur ve eşleşme sayısı artırılır. Bu işlem, tüm alternatif yollar keşfedilene kadar devam eder. Hopcroft-Karp algoritması, O(√V * E) zaman karmaşıklığı ile çalışır, burada V ve E, düğüm ve kenar sayısıdır.

### 5.2. Hungarian Algoritması (Kuhn-Munkres Algoritması):

Hungarian algoritması, iki parçalı graf üzerinde ağırlıklı maksimum eşleşmeyi bulmak için kullanılır. Bu algoritma, özellikle maliyet matrisi verildiğinde, atama problemlerini çözmek için uygundur. Algoritma, etiketleme ve eşleştirme adımlarını tekrarlayarak çalışır. Her adımda, etiketler güncellenir ve eşleştirme sayısı artırılır. Bu işlem, tüm eşleştirmeler gerçekleştirilene kadar devam eder. Hungarian algoritması, O(V^3) zaman karmaşıklığı ile çalışır, burada V grafın düğüm sayısıdır.

## Kullanılan Graf Algoritması: Dijkstra Algoritması

Dijkstra algoritması, tek kaynaklı en kısa yol algoritması olarak bilinir ve başlangıç düğümünden diğer tüm düğümlere en kısa yol uzunluklarını bulmak için kullanılır. Algoritma, pozitif kenar ağırlıklı grafiklerde en kısa yolu bulmak için tasarlanmıştır.

1956'da Edsger W. Dijkstra tarafından geliştirilen algoritma, başlangıç düğümünden hedef düğüme olan en kısa yolun ağırlığını ve rotasını bulur. Bu işlem, tüm düğümlerin uzaklıklarını tutan bir dizi ile yapılır ve başlangıç düğümünden hedef düğüme kadar olan en kısa yol, bu dizi üzerinden bulunur. Algoritma, her adımda en yakın düğümü seçer ve bu düğüme komşu olan düğümlerin uzaklıklarını günceller.

Dijkstra algoritması, bu projedeki dağıtım planı için uygun bir seçenek olmasına rağmen, başka algoritmalar da kullanılabilir. Örneğin, A* algoritması, özellikle büyük ölçekli grafiklerde daha hızlı sonuçlar veren, sezgisel bir yaklaşım sunar. Bu algoritma, Dijkstra algoritmasına benzer bir yaklaşım kullanarak, bir hedef düğüme kadar olan tahmini en kısa yol uzunluğunu hesaplar. Ancak A* algoritması, düğümler arasındaki mesafelerin yanı sıra, hedef düğüme olan tahmini mesafeyi de hesaba katar.

Floyd-Warshall algoritması ise, tüm çiftler arasındaki en kısa yolları bulmak için kullanılabilir. Bu algoritma, dinamik programlama prensibine dayanır ve tüm düğümler arasındaki en kısa yolların matrisini hesaplar. Bu matris, her iki düğüm arasındaki en kısa yol uzunluğunu içerir. Floyd-Warshall algoritması, özellikle grafikteki kenar ağırlıkları negatif değerler içeriyorsa veya birbirine doğrudan bağlı olmayan düğümler arasındaki en kısa yolları bulmak gerekiyorsa tercih edilir.

**Bu ödevde Dijkstra algoritması kullanılacak çünkü:**

Dijkstra algoritması, bu projede kullanılan graf algoritması olarak seçilmiştir ve bunun nedenleri aşağıdaki gibidir:

1. Algoritma, başlangıç düğümünden diğer tüm düğümlere en kısa yolları bulma konusunda etkilidir. Bu, dağıtım planı için önemli bir özelliktir, çünkü dağıtım ağının her bir noktasına en kısa sürede ulaşılması gerekmektedir. Dijkstra algoritması, başlangıç düğümünden hedef düğümlere olan en kısa yolları etkili bir şekilde hesaplar.

2. Pozitif ağırlıklı grafiklerle çalışır, bu da dağıtım planı için mesafe veya zaman gibi gerçek dünya ölçütlerine uygun olduğu anlamına gelir. Algoritma, gerçek dünya koşullarında işe yarayan pozitif ağırlıklı grafiklerle çalışır. Bu, mesafe ve zaman gibi gerçek dünya ölçütlerinin dağıtım planında kullanılabilmesi anlamına gelir.

3. Hesaplama karmaşıklığı, diğer algoritmalara kıyasla daha düşüktür (örneğin, Floyd-Warshall algoritması O(n^3) karmaşıklığına sahipken, Dijkstra algoritması O(n^2) karmaşıklığına sahiptir). Bu, daha hızlı hesaplamalar ve daha verimli dağıtım planları sağlar. Dijkstra algoritması, hesaplama karmaşıklığı açısından daha verimlidir. Özellikle, büyük ölçekli grafiklerde Floyd-Warshall algoritmasından daha hızlıdır. Bu, daha hızlı hesaplamalar ve daha verimli dağıtım planları için önemli bir avantaj sağlar.

4. Dijkstra algoritması, geniş bir uygulama yelpazesine sahip olduğundan, farklı dağıtım senaryolarına ve değişen ihtiyaçlara kolayca adapte olabilir. Bu, projenin gelecekteki genişlemeleri ve değişiklikleri için önemli bir esneklik sağlar. Örneğin, dağıtım planı daha büyük bir alanı kapsamaya başladığında veya farklı tedarikçiler veya müşteriler eklenirken, Dijkstra algoritması hala kullanılabilir ve uyarlanabilir.

Sonuç olarak, Dijkstra algoritması, bu projede kullanılan graf algoritması olarak seçildi, çünkü başlangıç düğümünden hedef düğümlere en kısa yolları etkili bir şekilde hesaplamakta, gerçek dünya ölçütleriyle çalışmakta ve hesaplama karmaşıklığı açısından daha verimli olmaktadır. Bu özellikler, projenin başarılı bir şekilde tamamlanması ve gelecekteki genişlemeler ve değişiklikler için gerekli esnekliğin sağlanması açısından önemlidir. Ayrıca, Dijkstra algoritması nispeten kolay anlaşılır ve uygulanabilir bir algoritma olduğu için bu projede kullanılacaktır.

Diğer yandan, projede kullanılacak başka graf algoritmaları da mevcuttur. Örneğin, A* algoritması, düğümler arasındaki tahmini en kısa mesafeyi hesaba katarak daha hızlı sonuçlar verir. Ancak bu algoritma, daha karmaşık bir yapıya sahip olduğu için, projenin gereksinimlerine uygun olmayabilir.

Floyd-Warshall algoritması ise, tüm çiftler arasındaki en kısa yolları bulmak için kullanılır. Bu algoritma, dağıtım planında çoklu rotaların kullanıldığı durumlarda faydalıdır. Ancak, hesaplama karmaşıklığı nedeniyle büyük ölçekli dağıtım ağlarında yavaş olabilir.

Sonuç olarak, Dijkstra algoritması, bu projede kullanılan graf algoritması olarak uygun bir seçimdir. Algoritmanın özellikleri, projenin gereksinimlerine uygun olması ve kolay uygulanabilir olması, projenin başarılı bir şekilde tamamlanmasını sağlamak için önemlidir. Bu algoritmanın seçilmesi, projeye esneklik ve uyumluluk sağlar, ekip üyelerinin çalışmalarını daha etkili bir şekilde sürdürmelerine ve projenin başarılı bir şekilde tamamlanmasına olanak tanır.

**Dijkstra algoritması, şu adımları içerir:**

**1.** Başlangıç düğümü seçilir ve bu düğümün en kısa yol mesafesi (kendi kendine) 0 olarak ayarlanır. Diğer tüm düğümlerin en kısa yol mesafeleri başlangıçta sonsuz olarak ayarlanır.

**2.** Tüm düğümler ziyaret edilmemiş düğüm kümesine eklenir.

**3.** Ziyaret edilmemiş düğümler arasında, en kısa yol mesafesine sahip olan düğüm seçilir (ilk adımda başlangıç düğümü seçilecektir).

**4.** Seçilen düğümün tüm komşuları için, mevcut en kısa yol mesafeleri, seçilen düğüme olan mesafe ve seçilen düğümden komşuya olan kenar ağırlığı toplamıyla karşılaştırılır. Eğer bu toplam, komşunun mevcut en kısa yol mesafesinden daha küçükse, komşunun en kısa yol mesafesi güncellenir.

**5.** Seçilen düğüm ziyaret edildi olarak işaretlenir ve ziyaret edilmemiş düğüm kümesinden çıkarılır.

**6.** Ziyaret edilmemiş düğüm kalmayana kadar 3-5 adımları tekrarlanır.

Bu adımların sonucunda, her düğümün en kısa yol mesafesi belirlenir ve başlangıç düğümünden diğer tüm düğümlere olan en kısa yollar hesaplanır. Bu hesaplama, pozitif kenar ağırlıklı grafiklerde en iyi sonucu verir.

Dijkstra algoritmasının önemli bir özelliği, hesaplama karmaşıklığının O(n^2) olmasıdır. Bu, algoritmanın büyük ölçekli grafiklerde yavaş çalışmasına neden olabilir. Ancak, bazı optimize edilmiş versiyonları (örneğin, min-heap veri yapısı ile) vardır ve bu versiyonlar, hesaplama karmaşıklığını O(n*log(n)) seviyesine indirerek performans iyileştirmeleri sağlayabilirler.

Sonuç olarak, Dijkstra algoritması, başlangıç düğümünden diğer tüm düğümlere en kısa yolu bulmak için kullanılan etkili bir algoritmadır. Algoritma, hesaplama karmaşıklığı açısından diğer algoritmalardan daha verimlidir ve pozitif kenar ağırlıklı grafiklerle çalışır. Bu nedenle, en kısa yol problemlerinin çözülmesinde yaygın olarak kullanılmaktadır.