T.C.

BURSA TEKNİK ÜNİVERSİTESİ

MÜHENDİSLİK VE DOĞA BİLİMLERİ FAKÜLTESİ

BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

Ali Doran, Mert Altın

1. 24360859211@ogr.btu.edu.tr/24360859211

2. 20360859061@ogr.btu.edu.tr/20360859061

RIP (ROUTİNG INFORMATİON PROTOCOL)

1.1 Giriş

RIP (Routing Information Protocol), ilk olarak 1970'lerde Xerox tarafından geliştirilen Gateway Information Protocol (GWINFO) üzerinden evrilmiştir. GWINFO, Xerox'un deneysel ağlarında yönlendirme bilgisi sağlamak amacıyla tasarlanmış bir protokoldü ve daha sonra Xerox Network Systems (XNS) protokol ailesine entegre edilerek XNS Routing Information Protocol (RIP) adıyla standartlaştırılmıştır (Zinin, 2002). XNS RIP, AppleTalk'un Routing Table Maintenance Protocol (RTMP) ve Novell'in IPX RIP gibi protokollerine de ilham vermiştir (Perlman R., 1999).

RIP, 1982 yılında BSD UNIX işletim sistemiyle birlikte açık kaynak olarak geniş çapta erişilebilir hale gelmiştir. Bu, protokolün popülerleşmesini sağlamış ve farklı ağ sistemlerinde kullanılmasını kolaylaştırmıştır (Hedrick, 1988). 1988 yılında Charles Hedrick tarafından yazılan bir dokümantasyonla RIP, RFC 1058 standardı altında RIPv1 olarak resmi bir protokol haline getirilmiştir (Hedrick C., 1988).

RIP'in temel aldığı Bellman-Ford algoritması, ilk olarak 1956'da Richard Bellman tarafından ve 1958'de Lester Ford Jr. tarafından bağımsız olarak tanımlanmıştır. Bellman-Ford algoritması, negatif ağırlıklı kenarların bulunduğu graflarda tek bir kaynaktan tüm diğer düğümlere en kısa yolları hesaplayabilme yeteneğiyle öne çıkar (Bellman, 1958). RIP, bu algoritmayı bir uzaklık-vektör yönlendirme yöntemi olarak kullanarak her yönlendiricinin komşularına düzenli olarak yönlendirme bilgisi göndermesi prensibiyle çalışır. Böylece ağdaki yönlendirme tabloları sürekli olarak güncel tutulur ve veri paketlerinin en uygun yoldan iletilmesi sağlanır (Perlman R., 1999).

Bellman-Ford algoritması, RIP'in bel kemiğini oluşturmuş ve bu sayede bilgisayar ağlarının erken dönemlerinde yönlendirme sorunlarının çözülmesinde önemli bir rol oynamıştır. RIP'in evrimi,

Xerox'un geliştirdiği deneysel protokollerden başlayarak modern IP tabanlı ağlara geçişin kritik bir parçası olmuştur (Zinin A., 2002).

RIP (Routing Information Protocol), küçük ve orta ölçekli ağlarda yönlendirme bilgilerini paylaşmak için kullanılan bir dinamik yönlendirme protokolüdür. Bu protokol, basitliği ve kolay uygulanabilirliği sayesinde özellikle karmaşık olmayan ağ topolojilerinde tercih edilmektedir (Bagcı T., 2023). Yönlendiriciler arasında otomatik bilgi alışverişini sağlayarak, ağ yönlendirme tablolarının manuel olarak güncellenmesi gerekliliğini ortadan kaldırır. Bu özellik, özellikle küçük işletmeler, eğitim kurumları ve daha basit ağ yapılarında büyük bir avantaj sağlar (Boyan F., 2023).

RIP'in temel avantajlarından biri, konfigürasyonunun kolay olmasıdır. Ancak, maksimum 15 atlama (hop) sınırı ve sınırlı metrik yapısı nedeniyle büyük ve karmaşık ağlarda verimsiz hale gelir. Daha büyük ağlarda genellikle OSPF veya EIGRP gibi daha gelişmiş protokoller tercih edilmektedir (Bilişim Evreni, 2023).

Sonuç olarak, RIP, basit ağ yapıları için tasarlanmış bir protokol olarak, az sayıda yönlendiriciye sahip olan ve karmaşıklığı düşük olan ağlarda başarılı bir şekilde kullanılmaktadır. Bu, ağ trafiğini yönlendirme sürecini daha az karmaşık hale getirerek hem zamandan hem de insan kaynağından tasarruf edilmesini sağlar (Erbaş C., 2019).

Router Information Protocol (RIP), küçük ve orta ölçekli ağlarda tercih edilen bir yönlendirme protokolüdür. Özellikle, ağ karmaşıklığının düşük olduğu ve güvenilir veri iletiminin önemli olduğu durumlarda yaygın olarak kullanılır. Protokol, yönlendiriciler arasında ağ topolojisi bilgisini paylaşarak veri paketlerinin en kısa yoldan hedefe ulaşmasını sağlar.

RIP'in basit yapısı ve kolay uygulanabilirliği, eğitim amaçlı ağ simülasyonlarında ve laboratuvar ortamlarında kullanılmasını destekler. Ayrıca, eski cihazlarla uyumlu olması ve minimal kaynak gereksinimi sayesinde, modern yönlendirme protokollerine geçiş yapamayan ağlarda hala tercih edilmektedir. Ancak, maksimum 15 atlama sınırı nedeniyle büyük ölçekli ağlarda kullanımı sınırlıdır (Kurose,J. F. & Ross K. W.,2017).

RIP, genellikle ağ geçitleri arasında temel yönlendirme bilgilerini paylaşmak için kullanılır. Bunun dışında, belirli güvenlik gereksinimlerini karşılamak adına statik yönlendirme ile birlikte kullanılarak esneklik sağlar.(Perlman R., 2000)

1.2 Teorik Temel

1.2.1 RIP Protokolünde Kullanılan Bellman-Ford Algoritması Matematiksel ve Teorik Temel

Bellman-Ford algoritması, bir başlangıç düğümünden diğer tüm düğümlere en kısa yolları bulan, yönlendirilmiş graflar üzerinde çalışan bir algoritmadır. Özellikle negatif ağırlıklı kenarların bulunduğu graflarda güvenilir bir çözüm sunması, bu algoritmayı Dijkstra gibi diğer algoritmalardan ayıran temel özelliklerden biridir. Bellman-Ford algoritması, dinamik programlama prensiplerini benimser ve bu yapısı sayesinde Routing Information Protocol (RIP) gibi uzaklık-vektör yönlendirme protokollerinde etkin bir şekilde kullanılır (Bellman, 1958).

1.2.2 Algoritmanın Temel Çalışma Prensibi

Bellman-Ford algoritması, bir grafın tüm kenarlarını iteratif olarak değerlendirerek, en kısa yol tahminlerini günceller. Bu süreç "relaksasyon" adı verilen bir işlemle gerçekleştirilir. Her iterasyonda bir kenar (u,v)(u, v)(u,v) üzerinde aşağıdaki işlem uygulanır:

d[v]=min(d[v],d[u]+w(u,v))

Bu formülde:

- d[v], kaynak düğümden v düğümüne olan mevcut en kısa yol tahminini ifade eder.
- d[u], kaynak düğümden u düğümüne olan mevcut en kısa yol tahminidir.
- w(u,v), u ve v düğümleri arasındaki kenarın ağırlığıdır.

Başlangıçta, kaynak düğümün mesafesi sıfır (d(s)=0d(s)) olarak atanırken, diğer tüm düğümlerin mesafesi sonsuz (∞) olarak belirlenir. Algoritma, grafın tüm kenarlarını |V|−1 kez (burada |V|, grafın düğüm sayısıdır) kontrol ederek her düğüm için en kısa yolu hesaplar (Cormen T. H., Leiserson, C. E., Rivest, R. L., & Stein, C., 2009).

1.2.3 RIP Protokolünde Kullanımı

RIP, Bellman-Ford algoritmasını temel alarak uzaklık-vektör yönlendirme protokolü şeklinde uygulanır. Protokol, yönlendiricilerin düzenli olarak komşularına kendi yönlendirme tablolarını göndermesiyle çalışır. Her yönlendirme tablosu, hedef ağlara olan mesafeleri ve

bu ağlara hangi yönlendirici üzerinden erişileceğini içerir. RIP protokolünde Bellman-Ford algoritmasının relaksasyon işlemi şu şekilde gerçekleşir:

$D_{yeni}(N) = \min(D_{yeni}(N), D_{gönderen}(N)+1)$

Burada:

- $D_{yeni}(N)$, yönlendiricinin hedef ağ N için mevcut mesafe tahminidir.
- $D_{gönderen}(N)$, komşu yönlendiriciden alınan hedef ağ N için mesafedir.
- +1, her yönlendirici arasındaki atlama (hop) sayısını ifade eder (Kurose J. F. & Ross K. W., 2013).

1.2.4 RIP Protokolünde Bellman-Ford Algoritmasının Veri Yapıları ve Kavramları

Bellman-Ford algoritması, temel olarak yönlendirilmiş graflar üzerinde çalışır ve bu grafların düğümleri ve kenarları ile ilişkilendirilmiş ağırlıkları üzerinden en kısa yolları hesaplar. Bu bağlamda kullanılan temel veri yapıları ve kavramlar aşağıda detaylandırılmıştır.

1.2.4.1 Grafik Yapısı

Bellman-Ford algoritmasının temelinde, yönlendirilmiş ve ağırlıklı bir graf yapısı bulunur. Bir graf, G=(V,E) şeklinde tanımlanır:

- **V:** Düğüm (veya yönlendirici) kümesi.
- **E:** Kenar (veya bağlantı) kümesi, her biri (u,v,w) şeklinde tanımlanır. Burada u ve v, kenarın bağladığı düğümleri ve w, kenarın ağırlığını temsil eder.

Graf, RIP protokolünde ağ topolojisini temsil eder. Düğümler yönlendiricileri, kenarlar ise yönlendiriciler arasındaki bağlantıları ifade eder. Ağırlık, genellikle bir yönlendiriciden diğerine olan atlama (hop) sayısını gösterir (Bellman R., (1958).

1.2.4.2 Dizi (Array) Yapısı

Bellman-Ford algoritmasında her düğümün kaynaktan olan en kısa yol tahminini saklamak için bir dizi (distance[]) kullanılır. Bu dizi:

- Başlangıçta tüm düğümler için sonsuz (∞) ile doldurulur.

- Kaynak düğüm için mesafe 0 olarak atanır (distance[source]=0).
- Algoritma ilerledikçe, dizi güncellenerek her düğüm için en kısa mesafe tahmini saklanır.

Ek olarak, bir predecessor[] dizisi de kullanılır. Bu dizi, her düğüm için en kısa yola ait bir önceki düğüm bilgisini saklar (Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C., 2009).

1.2.4.3 Kenar Listesi

Algoritma, grafın kenarlarını işlemek için genellikle bir kenar listesi veri yapısını kullanır. Bu liste, grafın tüm kenarlarını (u,v,w) formatında tutar:

- **u:** Kenarın başlangıç düğümü.
- **v:** Kenarın bitiş düğümü.
- **w:** Kenarın ağırlığı (örneğin, iki yönlendirici arasındaki atlama sayısı).

Kenar listesi, algoritmanın her iterasyonunda relaksasyon işlemi için kullanılır (Cormen T. H., Leiserson, C. E., Rivest, R. L., & Stein, C., 2009).

1.2.4.4 Kuyruk (Queue)

Bellman-Ford algoritması, genellikle bir kuyruk yapısı gerektirmez. Ancak RIP protokolündeki belirli durumlarda, yönlendirme bilgilerini yaymak için bir işleme kuyruğu kullanılabilir. Örneğin, yönlendirme tablolarındaki güncellemeler komşu yönlendiricilere sırayla iletilir ve bu sırayı yönetmek için kuyruklar kullanılabilir (Kurose J. F. & Ross K. W., 2013).

1.2.4.5 Relaksasyon Kavramı

Relaksasyon (gevşetme), Bellman-Ford algoritmasının ana işlevidir. Bir kenar (u,v,w) üzerinden geçerken, algoritma şu kontrolü yapar:

Eğer d[v]>d[u]+w(u,v) ise d[v]=d[u]+w(u,v)

Relaksasyon, algoritmanın her iterasyonunda daha kısa yolları bulmasını ve negatif döngüleri tespit etmesini sağlar (Cormen T. H., Leiserson, C. E., Rivest, R. L., & Stein, C., 2009).

1.2.4.6 Negatif Döngü Tespiti

Bellman-Ford algoritması, negatif ağırlıklı döngülerin varlığını tespit etme yeteneğine sahiptir. Algoritmanın son adımında, tüm kenarlar bir kez daha kontrol edilir. Eğer herhangi bir kenar (u,v,w) için d[v]>d[u] +w(u,v) koşulu sağlanıyorsa, graf negatif bir döngü içeriyor demektir (Cormen T. H., Leiserson, C. E., Rivest, R. L., & Stein, C., 2009).

1.2.4.7 Tablo (Routing Table) Kavramı

RIP protokolü, yönlendirme tabloları kullanarak Bellman-Ford algoritmasını uygular. Her yönlendirici, hedef ağlar için mesafe ve yön bilgilerini içeren bir tablo tutar. Bu tablolar:

- Belirli aralıklarla komşu yönlendiricilere gönderilir.
- Alınan bilgiler doğrultusunda güncellenir.
- Bellman-Ford algoritmasıyla en kısa yollar hesaplanarak optimize edilir (Hedrick C., 1988).

1.2.5 RIP Protokolünde Kullanılan Bellman-Ford Algoritmasının Avantajları ve Sınırlamaları

Bellman-Ford algoritması, sahip olduğu özelliklerle hem avantajlar hem de sınırlamalar sunan bir yapıya sahiptir. RIP protokolünde kullanılmasının ardındaki temel neden, küçük ve orta ölçekli ağlarda etkili bir yönlendirme çözümü sunmasıdır. Ancak, belirli kısıtlamaları nedeniyle daha büyük ve karmaşık ağlar için sınırlı bir kullanım alanı bulunmaktadır.

1.2.6 Avantajlar

- **Negatif Ağırlıklı Kenarları İşleyebilme Yeteneği**

  Bellman-Ford algoritması, negatif ağırlıklı kenarların bulunduğu graflarda güvenilir bir şekilde çalışabilir. Bu özellik, RIP gibi uzaklık-vektör protokolleri için kritik bir avantaj sağlar çünkü algoritma, negatif ağırlıklı döngülerin varlığını tespit edebilir (Bellman, 1958).

- **Dinamik Güncelleme Yeteneği**

  Algoritma, ağda meydana gelen değişikliklere dinamik olarak uyum sağlayabilir. Örneğin, bir düğüm veya bağlantının devre dışı kalması durumunda yönlendirme tabloları otomatik olarak güncellenir (Hedrick, 1988).

- **Basit ve Kolay Uygulanabilir Yapı**

  Bellman-Ford algoritması ve dolayısıyla RIP protokolü, nispeten basit bir yapıya sahiptir. Bu durum, ağ yöneticileri tarafından kolayca konfigüre edilmesini ve uygulanmasını sağlar (Kurose ve Ross, 2013).

- **Yaygın Uyumluluk**

  RIP protokolü, eski ağ cihazlarıyla uyumluluk sağlar ve Bellman-Ford algoritmasının basitliği bu protokolün geniş bir donanım yelpazesinde çalışmasını mümkün kılar (Perlman, 1999).

1.2.7 Sınırlamalar

- **Zaman Karmaşıklığı**

  Bellman-Ford algoritmasının zaman karmaşıklığı O(|V|·|E|)O(|V| \cdot |E|)O(|V|·|E|)'dir. Bu, büyük ağlarda performans problemlerine yol açabilir. Bu nedenle algoritma, genellikle küçük ve orta ölçekli ağlar için uygun görülür (Cormen, Leiserson, Rivest ve Stein, 2009).

- **Sayıya Kadar Sayma Problemi (Count to Infinity)**

  RIP protokolünde, bir ağ segmenti devre dışı kaldığında bu bilgi ağdaki yönlendiricilere yayılana kadar uzun bir süre geçebilir. Bu durum, "sayıya kadar sayma" olarak bilinen bir problem oluşturur ve ağın stabilizasyonunu geciktirir (Hedrick, 1988).

- **Maksimum Atlama Sınırı**

  RIP protokolünde mesafe metriği olarak atlama (hop) sayısı kullanılır ve bu değer 15 ile sınırlıdır. 16 atlama, "sonsuz" olarak kabul edilir. Bu sınırlama, protokolün büyük ve karmaşık ağlarda kullanımını engeller (Kurose ve Ross, 2013).

- **Hatalara Karşı Hassasiyet**

RIP, aldığı bilgileri körü körüne kabul eder. Yanlış bir yönlendirme tablosu veya bozulmuş veri, tüm ağın işleyişini etkileyebilir. Bu durum, protokolün güvenilirliğini azaltır (Perlman, 1999).

- **Verimlilik Eksikliği**

  Bellman-Ford algoritması, diğer modern yönlendirme protokollerine kıyasla daha fazla ağ trafiği yaratabilir. Örneğin, RIP'in yönlendirme bilgilerini düzenli olarak göndermesi (her 30 saniyede bir) ağ kaynaklarının verimli kullanımını sınırlar (Hedrick, 1988).

1.3 Algoritmanın Uygulama Alanları

1.3.1 Algoritma: Girdi, İşleme Süreci ve Çıktı

1.3.1.1 Girdi

Tanım: Algoritmanın başlangıçta çalışabilmesi için aşağıdaki girdilere ihtiyaç vardır:

- **Düğüm Sayısı**: Grafın toplam düğüm sayısını ifade eder.
- **Kenar Listesi**: Grafın kenarlarını içerir. Her kenar, üç bilgiyle tanımlanır: başlangıç düğümü, bitiş düğümü ve kenar ağırlığı. Ağırlıklar genellikle mesafeyi veya maliyeti ifade eder.
- **Kaynak Düğüm**: En kısa yolların hesaplanacağı başlangıç düğümüdür.

```
class Grafik:

    def __init__(self, dugum_sayisi):

        self.dugum_sayisi = dugum_sayisi  # Düğüm sayısı

        self.kenarlar = []                # Kenar listesi


    def kenar_ekle(self, baslangic, bitis, agirlik):

        """

        Kenar ekleme: başlangıç -> bitiş (ağırlık)

        """

       self.kenarlar.append((baslangic, bitis, agirlik))
```

Açıklama

- **dugum_sayisi:** Grafın düğüm sayısını belirler.
- **kenar_ekle:** Her çağrıldığında bir kenar eklenir. Bu bilgi, grafın topolojisini tanımlar.

1.3.1.2 İşleme Süreci

Bellman-Ford algoritması, işleme sürecinde üç ana adım izler:

a. Başlatma: Algoritma, başlangıçta tüm düğümler için mesafeleri sonsuz (∞\infty∞) olarak belirler. Ancak kaynak düğümün mesafesi sıfırdır. "Önceki düğüm" bilgisi, yolları geri takip etmek için başlatılır.

```
mesafeler = [float('inf')] * self.dugum_sayisi

mesafeler[kaynak] = 0

onceki_dugumler = [-1] * self.dugum_sayisi
```

Açıklama

- **mesafeler:** Her düğüm için mevcut en kısa mesafeyi tutar.
- **onceki_dugumler**: Her düğüm için en kısa yol üzerinde bir önceki düğümü kaydeder.

b. Kenarların Gevşetilmesi: Grafın tüm kenarları üzerinde V−1 kez iterasyon yapılır. Her kenar için, eğer başlangıç düğümünden geçerek bir düğüme daha kısa bir yol bulunursa, mesafeler güncellenir. RIP protokolüne uygun olarak, maksimum 15 atlama sınırı uygulanır.

```
for _ in range(self.dugum_sayisi - 1):

    for baslangic, bitis, agirlik in self.kenarlar:

        if mesafeler[baslangic] != float('inf') and \

                    mesafeler[baslangic]  +  agirlik  <
mesafeler[bitis]:

            yeni_mesafe = mesafeler[baslangic] + agirlik

             if yeni_mesafe <= 15:  # Maksimum 15 atlama
sınırı

        mesafeler[bitis] = yeni_mesafe

                onceki_dugumler[bitis] = baslangic
```

Açıklama:

- **Gevşetme İşlemi**: Mevcut mesafe, daha kısa bir yol bulunursa güncellenir.
- **Maksimum Atlama Kontrolü**: RIP protokolünün sınırlarına uygun olarak 15 atlamayı aşan yollar güncellenmez.

c. Negatif Döngü Kontrolü: Negatif ağırlıklı döngülerin varlığını tespit etmek için tüm kenarlar bir kez daha kontrol edilir. Eğer daha kısa bir yol bulunursa, bu durum negatif döngünün varlığına işaret eder.

```
for baslangic, bitis, agirlik in self.kenarlar:
    if mesafeler[baslangic] != float('inf') and \
       mesafeler[baslangic] + agirlik < mesafeler[bitis]:
       raise ValueError("Grafik negatif ağırlıklı döngü
içeriyor!")
```

Açıklama

- Negatif ağırlıklı döngüler, ağın güvenliğini ve işlevselliğini etkileyebilir. Bu durum tespit edildiğinde algoritma hata verir.

1.3.1.3 Çıktı

Tanım: Çıktılar, algoritmanın sonuçlarını ifade eder:

- **Mesafeler**: Kaynak düğümden diğer düğümlere olan en kısa mesafeler.
- **Yollar**: Her düğüm için kaynak düğümden başlayarak en kısa yol üzerinde gidilen düğümler.

```
mesafeler = [mesafe if mesafe <= 15 else float('inf') for
mesafe in mesafeler]
return mesafeler, onceki_dugumler
```

Açıklama

- **"Ulaşılamaz" Mesafeler**: 15'i aşan mesafeler ulaşılmaz olarak işaretlenir.
- **Tam Yol Çıktısı**: onceki_dugumler dizisi, her düğüm için en kısa yolu temsil eden düğümleri sıralar.

1.3.1.4 Çıktıyı Yazdırma

```
def en_kisa_yolu_yazdir(self, kaynak, mesafeler, onceki_-
dugumler):

    print(f"Kaynak Düğüm: {kaynak}")

    print("Düğüm   Mesafe   Yol")

    for hedef in range(self.dugum_sayisi):

        if mesafeler[hedef] == float('inf'):

            print(f"{hedef}\t\tUlaşılamaz")

        else:

            yol = []

            temp = hedef

            while temp != -1:

                yol.append(temp)

                temp = onceki_dugumler[temp]

            yol.reverse()

          print(f"{hedef}\t\t{mesafeler[hedef]}\t\t{' ->
'.join(map(str, yol))}")
```

1.3.2 Algoritmanın Hangi Sektörlerde veya Problemlerde Kullanıldığı

- **Ağ Yönlendirme**: RIP protokolü gibi yönlendirme protokollerinde, veri paketlerinin en kısa yollardan yönlendirilmesi.

- **Telekomünikasyon**: Veri yollarını optimize ederek ağ trafiğini düzenlemek.
- **Ulaşım ve Lojistik**: Şehirlerarası taşımacılıkta en kısa yolların bulunması.
- **Enerji Dağıtımı**: Elektrik hatlarında enerji kaybını en aza indiren yolların bulunması.
- **Yapay Zeka ve Robotik**: Robotların hareket planlaması ve rota optimizasyonu.

1.4 Performans Analizi

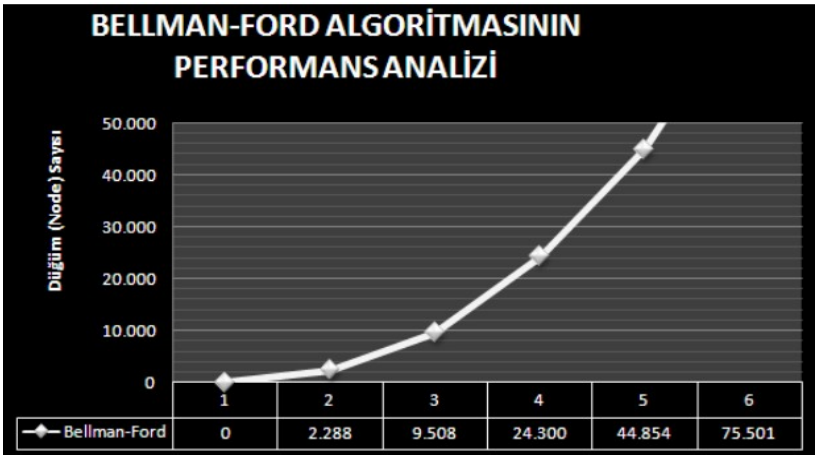
1.4.1 En Kötü Durum Zaman Karmaşıklığı Hesaplama Süreci

- **Başlatma Adımı**: Algoritmanın başında tüm düğümlerin mesafelerini ∞ olarak ayarlıyoruz. Bu işlem O(V) zaman alır. (V = düğüm sayısı)
- **Kenarların Gevşetilmesi**: Algoritma, tüm kenarları $V-1$ kez işler. Her kenar gevşetme işlemi sabit zaman alır. Eğer grafın E kenarı varsa, bu adım için toplam süre $O(V \cdot E)$ olur (Optimizing Bellman-Ford Algorithm, n.d.).
- **Negatif Döngü Kontrolü**: Ek bir iterasyon yapılarak tüm kenarlar tekrar kontrol edilir. Bu işlem O(E) zaman alır.

Sonuç olarak, Bellman-Ford algoritmasının **toplam zaman karmaşıklığı**: $O(V \cdot E)$

1.4.2 Ortalama Durum Zaman Karmaşıklığı

Ortalama durumda, grafın yoğunluğuna (kenar sayısına) bağlı olarak performans değişir. Ancak Bellman-Ford algoritması, her durumda kenarları $V-1$ kez işlemeyi içerdiği için ortalama karmaşıklık da $O(V \cdot E)$'dir.



Şekil 1. *Bellman Ford Algoritması Uygulaması ve Çalışma Hızı*

1.4.3 Uzay Karmaşıklığı

Bellman-Ford algoritması, birkaç temel veri yapısını saklamak için bellek kullanır:

- **Mesafeler Dizisi (**mesafeler**)**: Her düğüm için bir mesafe değeri saklar. Bu, O(V)bellek gerektirir.
- **Önceki Düğümler Dizisi (**onceki_dugumler**)**: Her düğüm için bir önceki düğüm bilgisini saklar. Bu da O(V)bellek gerektirir.
- **Kenar Listesi (**self.kenarlar**)**: Grafın kenarlarını saklar. Eğer grafın E kenarı varsa, bu O(E)bellek gerektirir.

Sonuç olarak, Bellman-Ford algoritmasının **toplam uzay karmaşıklığı**: O(V+E)

1.4.4 Bellman-Ford Algoritmasının Geliştirme ve Optimizasyon Yöntemleri

Bellman-Ford algoritması, negatif ağırlıklı kenarların bulunduğu grafiklerde en kısa yolları bulmak için etkili bir yöntemdir. Ancak, zaman karmaşıklığının O(V·E) olması, özellikle büyük ve yoğun grafiklerde performans sınırlamalarına yol açabilir. Bu nedenle, algoritmanın verimliliğini artırmak amacıyla çeşitli iyileştirme ve optimizasyon teknikleri geliştirilmiştir.

- **Erken Sonlandırma (Early Termination):** Algoritma, her iterasyonda tüm kenarları gevşetir. Eğer bir iterasyon sırasında hiçbir mesafe güncellenmezse, algoritma en kısa yolları bulmuştur ve daha fazla iterasyona gerek yoktur. Bu yaklaşım, gereksiz iterasyonları önleyerek ortalama çalışma süresini azaltır (Optimizing Bellman-Ford Algorithm, n.d.).
- **Kenar Listesinin Dinamik Güncellenmesi:** Her iterasyonda, sadece önceki iterasyonda mesafesi güncellenen düğümlerin komşu kenarlarını gevşetmek, işlem sayısını azaltır. Bu yöntem, özellikle yoğun grafiklerde performansı artırır (Optimizing Bellman-Ford Algorithm, n.d.).
- **Yen'in İyileştirmesi:** Yen (1970), düğümleri belirli bir sıraya göre işleyerek ve kenarları iki alt kümeye ayırarak, her iterasyonda en az iki kenarın doğru mesafeye ulaşmasını sağlar. Bu, toplam iterasyon sayısını yarıya indirir (Dijkstra ve Bellman-Ford En Kısa Yol Algoritmalarının Karşılaştırılması, 2002).
- **Rastgele Permütasyon ile İyileştirme:** Düğüm sıralamasını rastgele permütasyonlarla belirlemek, en kötü durum senaryolarının olasılığını azaltır ve ortalama performansı iyileştirir (Optimizing Bellman-Ford Algorithm, n.d.).
- **Paralel İşleme:** Algoritmanın paralel versiyonları, özellikle büyük grafiklerde, birden fazla işlemci kullanarak çalışma süresini önemli ölçüde azaltabilir (Optimizing Bellman-Ford Algorithm, n.d.).

1.4.4.1 Alternatif Algoritmalar

Bellman-Ford algoritmasının performansını artırmak veya belirli durumlarda daha verimli çözümler sunmak amacıyla çeşitli alternatif algoritmalar geliştirilmiştir:

- **Dijkstra Algoritması:** Pozitif ağırlıklı kenarlara sahip grafiklerde, Dijkstra algoritması genellikle daha hızlıdır ve zaman karmaşıklığı O(V2) veya uygun veri yapılarıyla O(E+VlogV) olabilir. Ancak, negatif ağırlıklı kenarların bulunduğu grafiklerde güvenilir sonuçlar vermez (Dijkstra ve Bellman-Ford En Kısa Yol Algoritmalarının Karşılaştırılması, 2002).

- **A (A-Star) Algoritması:** Heuristik yöntemler kullanarak, hedefe yönelik arama yapar ve özellikle yol bulma problemlerinde etkilidir. Ancak, doğru ve uygun bir sezgisel fonksiyon gerektirir (Bellman–Ford algorithm, 2023).
- **Johnson Algoritması:** Tüm çiftler arasındaki en kısa yolları bulmak için kullanılır ve negatif ağırlıklı kenarları olan grafiklerde etkilidir. Zaman karmaşıklığı $O(V2\log V+V\cdot E)$ olup, büyük ve seyrek grafiklerde avantajlıdır (Dijkstra ve Bellman-Ford En Kısa Yol Algoritmalarının Karşılaştırılması, 2002).
- **Delta-Stepping Algoritması:** Paralel işleme için tasarlanmış bu algoritma, özellikle büyük ölçekli grafiklerde ve pozitif ağırlıklı kenarlarla çalışırken etkilidir (Bellman–Ford algorithm, 2023).

Bu iyileştirme yöntemleri ve alternatif algoritmalar, Bellman-Ford algoritmasının sınırlamalarını aşmak ve belirli uygulama alanlarında daha verimli çözümler sunmak amacıyla geliştirilmiştir.

1.5 Çalışma Soruları ve Egzersizler

En başından beri RİP protokolünün "Belman Ford" algoritmasını kullandığını ve bunlarla ilgi detaylı açıklamaları yaptık. Şimdi de algoritmanın nasıl çalıştığını ve ilgili örnekler üzerinde inceleyelim.

1.5.1 Algoritmanın Temel Çalışma Prensibi

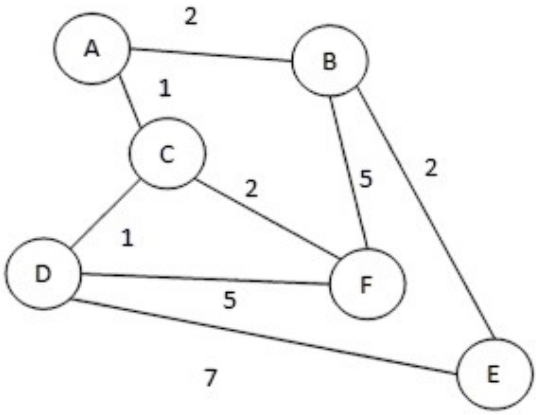
- Her bir router bir düğüm olarak düşünürsek bütün düğümlere sonsuz ifadesini ata
- Bir başlangıç düğümü belirle ve harf veya simge ata (r1 diyelim)
- r1 başlangıç düğümü olduğundan buraya gelme maliyetimiz 0 olarak belirlenir.
- Daha sonra dolaşacağımız tüm routerlar belirlenir ve bunlara da bir sıra atanır. Biz örneklerin anlaşılır olabilmesi için sıraya uygun olarak yerleştirmiş olacağız fakat bu durum tamamen rastgele olur.
- Tüm routerlar belirlendikten sonra bu routerler arasındaki tüm vağlantılar belirlenir ve maliyetleri yazılır.(Belman Ford algoritması tüm kenarları teker teker dolaşması sebebiyle diğer

benzer algoritmalardan ayrılır ve bu sebeple routerlar da kullanılır.)

- Belirtilen bu maliyetlere göre her düğümden diğer bir düğüme geçmek için en kısa yollar bulunarak bir tablo haline getirilir ve bu tablo tüm routerlarda bulunur.

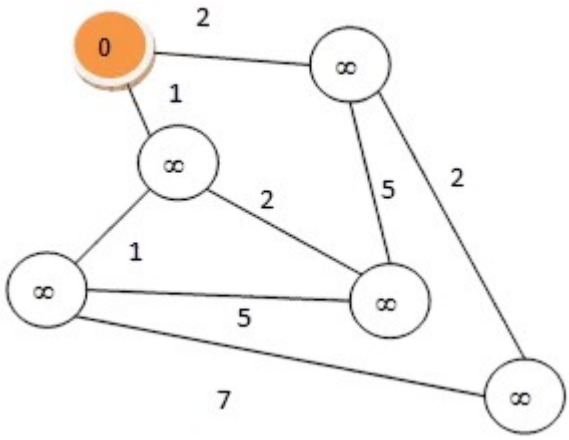
Örnek 1:

Belman Ford algoritması mantığını anlamak için başlangıç olarak kolay bir örnekle başlayalım;
Elimizde 6 düğüm ve 8 kenardan oluşan bir yapımız olsun;



Şekil 2. *Örenk 1 için taslak*

Ayrıca başlangıç düğümü A, hedef düğüm olarak da E düğümünü tanımlayalım . Bu düğümleri algoritmanın sonunda kontrol için kullanacağız.
A başlangıç olduğu için 0 değerine sahiptir.



Şekil 3. *Örnek 1 için ilk adım*

Önce dolaşacağımız düğümler için bir sıra oluşturalım. Düğüm sıramız: A, B, C, D, E, F olsun.

Şimdi de kenarları sıralayalım:

AB=2

AC=1

BE=2

BF=5

CF=2

CD=1

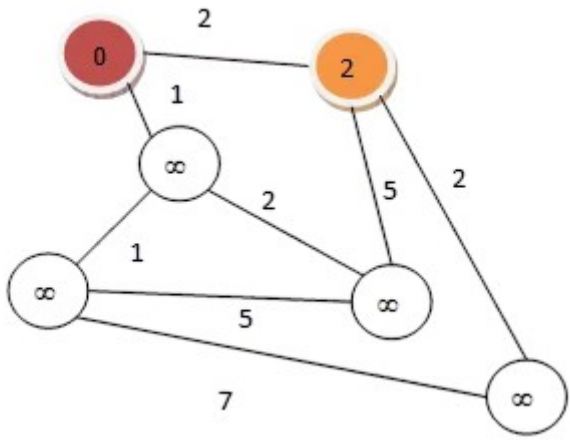
DF=5

DE=7

Sırayla yukarıdaki kenarları (edges) dolaşır ve grafikteki değerleri günceller. Algoritma öncelikli olarak düğümleri dolaşıyor ve her düğüm için kenarları dolaşacak.

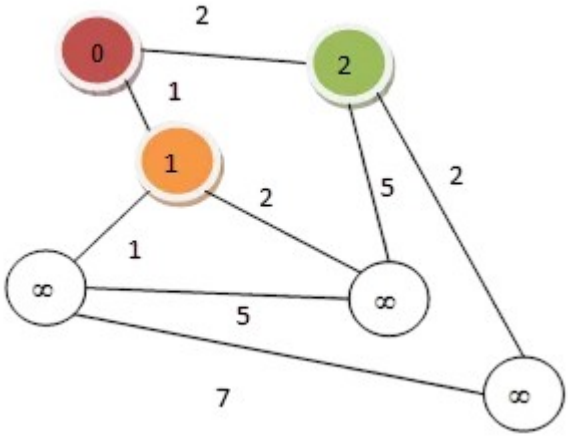
A düğümü için kenarları dolaşıyoruz. Kenarlardan sadece AB ve AC kenarları, A düğümü ile ilgili. Aslında bütün kenarlar dolaşılıyor olmasına rağmen, sadece bu iki kenar, graftaki sonucu etkileyecek.

AB 2, min (A, B) = 0 --> 0+ 2 = 2



Şekil 4. *Örnek 1 için ikinci adım*

AC 1, min (A, C) = 0 => 0+1 = 1



Şekil 5. *Örnek 1 için üçüncü adım*

Ardından B düğümüne geçiyoruz. Ve bu düğüm için bütün kenarları tekrar dolaşmaya başlıyoruz.

Bu durumda etkisi görülecek kenarlar:

AB=2

AC=1

BE=2

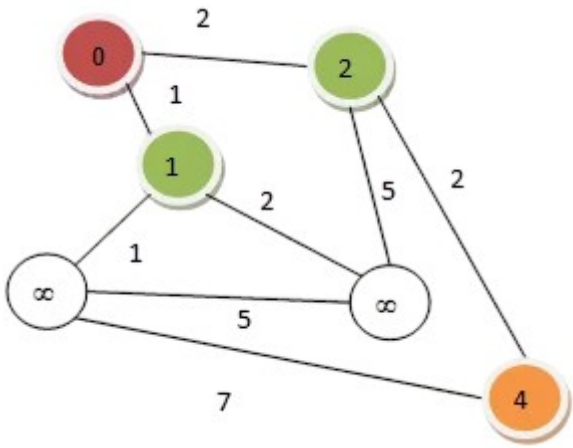
BF=5

CF=2

CD=1

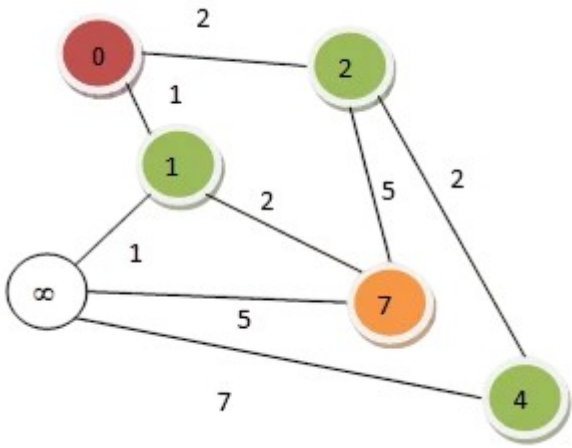
Kenarlarıdır çünkü bunlar dışındaki kenarların bağladıkları düğümler sonsuz değerindedir ve güncelleme olmaz. Diğer bir deyişle grafın yukarıdaki şeklinde, A, B, C düğümlerinde sonsuz dışında değerler bulunuyor. O halde sadece bu düğümlere komşu olan kenarları almak yeterlidir.

BE 2, min (B, E) 2, => 2 + 2 = 4



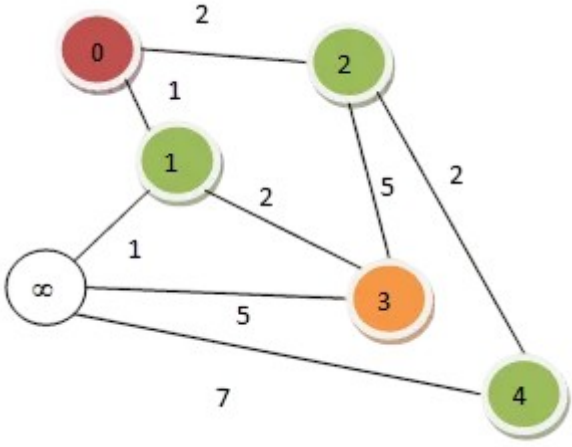
Şekil 6. *Örnek 1 için dördüncü adım*

BF 5, min (B, F) = 2 => 2+5 = 7



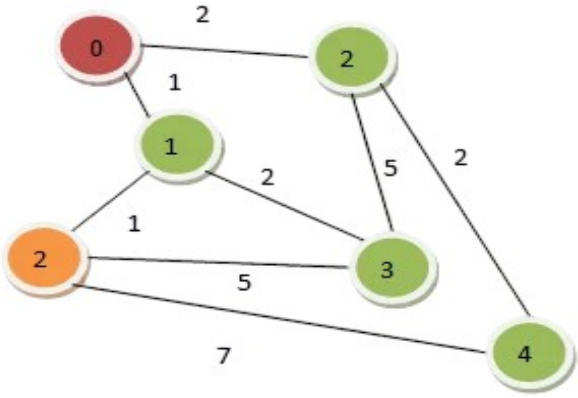
Şekil 7. *Örnek 1 için beşinci adım*

CF 2, min (C, F) = 1 => 1+2 = 3, bu değer 7'den küçük olduğu için güncelliyoruz:



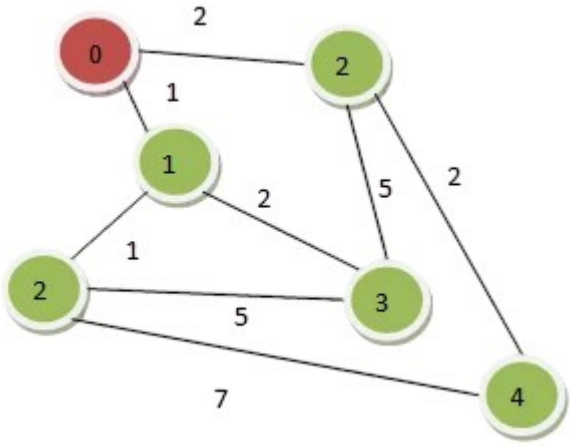
Şekil 8. *Örnek 1 için altıncı adım*

CD -1, min (C, D) = 1=> 1+(-1) = 0



Şekil 9. *Örnek 1 için yedinci adım*

Yukarıdaki son halimizde henüz 2 düğüm için işlem yapıldı (A ve B düğümleri).



Şekil 10. *Örnek 1 için sekizinci adım*

Graf 2 düğüm için kararlı hale ulaşmıştır ve diğer düğümler için işlem devam etse bile graftaki değerler değişmez. Böylelikle istediğimiz

herhangi bir düğüme gitmek için A (yani 0) başlangıç konumu olmak üzere gidilmek istenilen düğümün içindeki sayı en kısa yol olmuş oluyor. Tabi bu düğümleri bir router olarak düşüneceğiz bu sebeple her router bu mesafeleri bilmek zorunda bu sebeple bir tablo oluşturmamız gerekir. Tablomuz hedef, gecikme ve sonraki düğüm olmak üzere 3 kısımdan oluşur.

RİP protokolüne göre de tablo oluşturma:

- **Hedef:** Üsteki örnek için A düğümünden başlanıp hangi düğüme gitmek isteniyor.
- **Gecikme:** Hedefe gidilirken ki maliyetim
- **Sonraki Düğüm:** Hedefe varmadan önceki üzerinden geçilen düğümdür. Eğer Başlangıç düğümünde isek yoktur, herhangi bir düğüme uğramadan hedefe ulaşıyor isek hedef düğümün kendisidir.

Sadece A routerının tutacağı mesafe tablosu şu şekildedir;

Tablo 1. *Örnek 1 için A routerının sakladığı tablo*

HEDEF	GECİKME	SONRAKİ DÜĞÜM
A	0	-
B	2	B
C	1	C
D	2	C
E	4	B
F	3	C

Örnek 2:

Yukarıdaki örnekte, düğümlerin ve kenarların rast gele dizildiğinden bahsetmiştik. Acaba bu dizilim rast gele olarak yukarıdakinden farklı olsaydı Bellman-Ford yine başarılı çalışır mıydı?
Yukarıdaki dizilimi tersine çevirelim:

DF=5

DE=7

CF=2

CD=1

BF=5

BE=2

AC=1

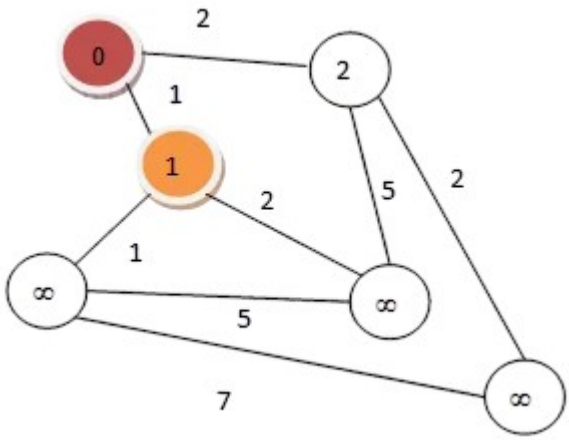
AB=2

Bu yeni dizilime göre algoritmamızı çalıştırıyoruz:

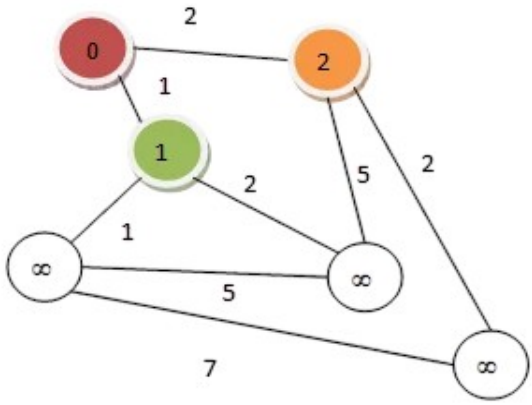
A düğümü için kenarları dolaşıyoruz. Kenarlardan sadece AB ve AC kenarları, A düğümü ile ilgili. Aslında bütün kenarlar dolaşılıyor olmasına rağmen, sadece bu iki kenar, graftaki sonucu etkileyecek.

AC 1, min (A, C) = 0 => 0+1 = 1

AB 2, min (A, B) = 0 --> 0+ 2=2



Şekil 11. *Örnek 2 için ilk adım*



Şekil 12. *Örnek 2 için ikinci adım*

Ardından B düğümüne geçiyoruz. Ve bu düğüm için bütün kenarları tekrar dolaşmaya başlıyoruz.

Bu durumda etkisi görülecek kenarlar:

CF=2

CD=1

BF=5

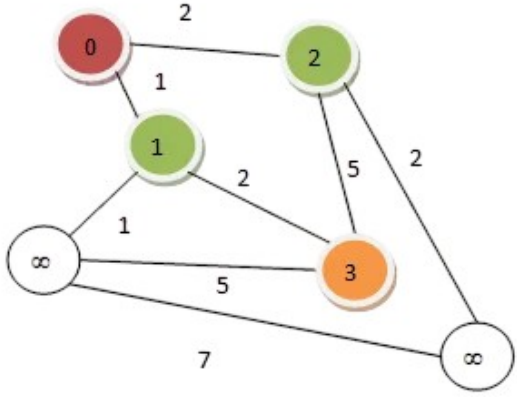
BE=2

AC=1

AB=2

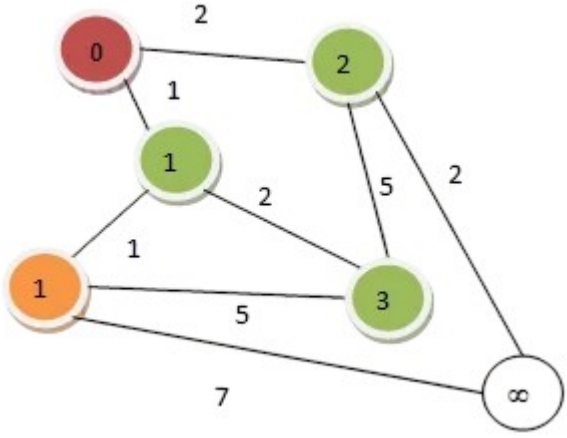
Kenarlarıdır çünkü bunlar dışındaki kenarların bağladıkları düğümler sonsuz değerindedir ve güncelleme olmaz. Diğer bir deyişle grafın yukarıdaki şeklinde, A, B, C düğümlerinde sonsuz dışında değerler bulunuyor. O halde sadece bu düğümlere komşu olan kenarları almak yeterlidir.

CF 2 için:



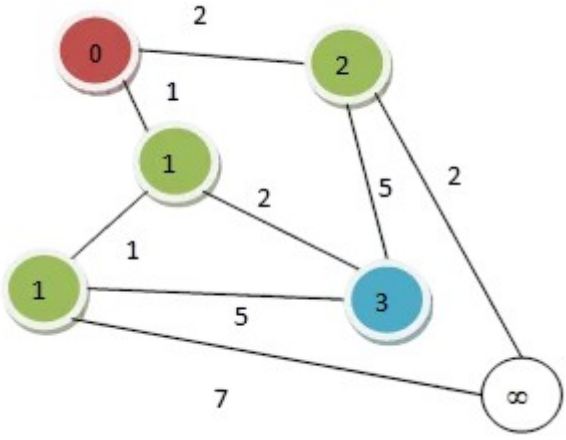
Şekil 13. *Örnek 2 için üçüncü adım*

CD 1 için**:**



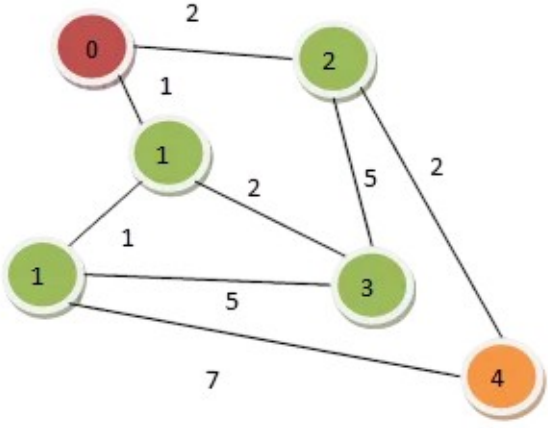
Şekil 14. *Örnek 2 için dördüncü adım*

BF 5 için, zaten daha kısa olan 3 değeri bulunmuştur**.**



Şekil 15. *Örnek 2 için beşinci adım*

BE 2 için**:**



Şekil 16. *Örnek 2 için altıncı adım*

Tablomuz ;

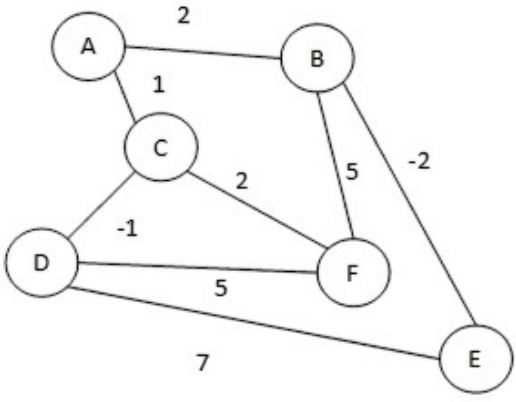
Tablo 2. *Örnek 2 için A routerının sakladığı tablo*

HEDEF	GECİKME	SONRAKİ DÜĞÜM
A	0	-
B	2	B
C	1	C
D	2	C
E	4	B
F	3	C

AC ve AB kenarları için ise değişiklik olmaz. Yukarıdaki bu yeni çalışmada dikkat edilirse, bir önceki çalışmadan farklı olarak F düğümü hiç 7 olmadan 3 değerini bulmuştur. Görüldüğü üzere kenar sıralaması, sonucu etkilememekle birlikte çalışma hızını etkileyebilir.

Örnek 3:

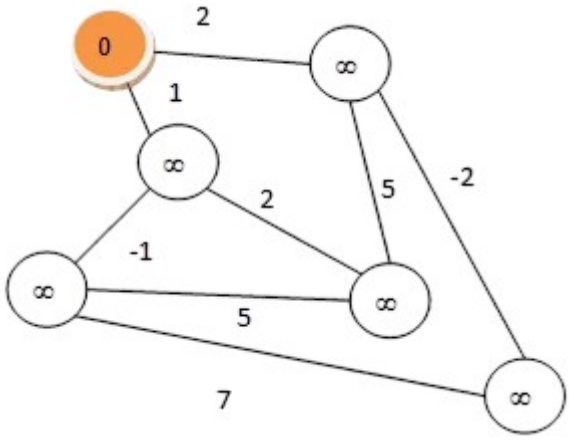
Şimdiye kadar çözdüğümüz örnekler hep pozitif değerler içindi. Şimdide negatif değerler için bir örnek çözelim.
Eksi değer olması durumuna elimizde böyle bir graf olsun;



Şekil 17. *Örnek 3 için taslak*

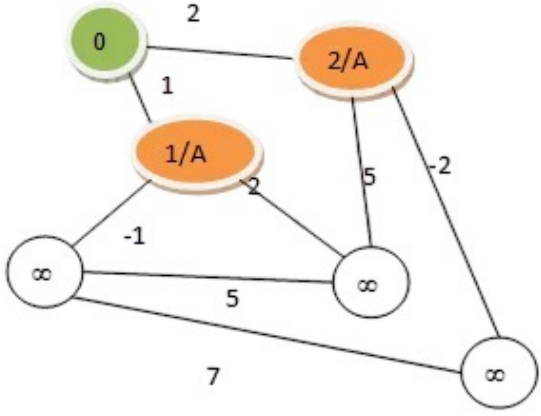
Yukarıdaki şekilde görüldüğü üzere, BE ve CD kenarları eksi değerlidir. Bu şeklin çalışmasını inceleyelim:

Her şey, normal algoritmada olduğu gibi, başlangıç dışındaki düğümlere sonsuz atayarak başlıyor.



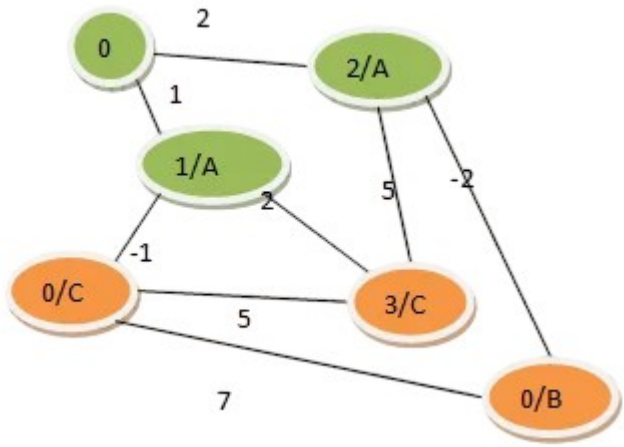
Şekil 18. *Örnek 3 için ilk adım*

Ardından her düğüm için ve her kenar için işlem yapılıyor. Bu adımları daha önce detaylıca gösterdiğim için aşağıda düğüm bazlı hızlandırılmış olarak anlatıp, sadece eksi değerdeki farkı vurgulayacağım:



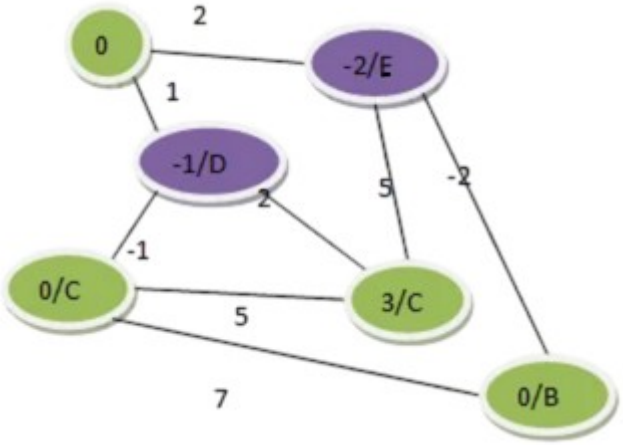
Şekil 19. *Örnek 3 için ikinci adım*

Bir sonraki adımda kenar sırasına göre aşağıdaki güncellemeler yapılır:



Şekil 20. *Örnek 3 için üçüncü adım*

Bu haliyle, graftaki bütün düğümlere erişilmiştir. Fakat problem bitmez. Yani eksi değerli düğümler, çalışmanın hala devam etmesini gerektirir.



Şekil 21. *Örnek 3 için dördüncü adım*

Yukarıdaki son hali, kararlı haldir.

Tablomuz ;

Tablo 3. *Örnek 3 için A routerının sakladığı tablo*

HEDEF	GECİKME	SONRAKİ DÜĞÜM
A	0	-
B	-2	E
C	-1	D
D	0	C
E	0	B
F	3	C

Bu noktada şu soru sorulabilir: Peki neden tekrar eski değerli düğümler işlenerek, D ve E düğümleri güncellemiyor?

Bu soru haklı bir sorudur ancak yazının başında belirtildiği üzere, Bellman-Ford algoritması, her düğümde, kimden gelindiği bilgisini tutar. Dolayısıyla D düğümünü 0 olarak güncellemeden önce, D düğümüne, C düğümünden gelindiği ve E düğümüne de B düğümünden gelindiği kaydedilmiştir. Dolayısıyla, eksi değerler tağılan bu döngülerde, tekrar edilip örneğin F düğümüne -4 değeri yazılamaz. Çünkü F düğümü zaten B düğümünden güncellenmiştir.

Algoritma Özeti

RIP protokolü, ağ yönlendirmesi için Bellman-Ford algoritmasını temel alarak çalışır ve her yönlendiricinin düzenli olarak komşularına yönlendirme tablolarını göndererek dinamik bir yapı sunar. Algoritma, negatif ağırlıklı kenarları işleyebilme kabiliyeti sayesinde güvenilir bir şekilde en kısa yolları bulur. Ancak, RIP protokolünün 15 atlama sınırı ve Bellman-Ford algoritmasının "count to infinity" gibi problemleri, protokolün büyük ve karmaşık ağlarda kullanımını sınırlandırır. Küçük ve orta ölçekli ağlarda etkili bir çözüm sunan RIP, daha büyük ağlar için OSPF gibi alternatif protokollerin geliştirilmesine zemin hazırlamıştır.

Kaynakça

- Bellman, R. (1958). On a routing problem. **Quarterly of Applied Mathematics, 16**(1), 87-90. https://doi.org/10.1090/qam/102435
- Hedrick, C. (1988). Routing Information Protocol. **RFC 1058**. Retrieved from https://www.rfc-editor.org/rfc/rfc1058.html
- Perlman, R. (1999). **Interconnections: Bridges, Routers, Switches, and Internetworking Protocols**. Addison-Wesley. https://www.pearson.com/store/p/interconnections/P100000158034
- Zinin, A. (2002). **Cisco IP Routing: Packet Forwarding and Intra-domain Routing Protocols**. Addison-Wesley. https://www.ciscopress.com/store/cisco-ip-routing-packet-forwarding-and-intra-domain-9781587050063
- Bagcı, T. (2023). **RIP nedir ve nasıl çalışır?** Tolga Bağcı. Retrieved from https://www.tolgabagci.com/rip-nedir/
- Boyan, F. (2023). **Dinamik Yönlendirme Protokolü: RIP**. Fırat Boyan Blog. Retrieved from https://www.firatboyan.com/dynamic-routing-dinamik-yonlendirme-protokolu-rip.aspx
- Bilişim Evreni. (2023). **RIP Protokolü Nedir?**. Retrieved from https://bilisimevreni.com.tr/rip-protokolu-nedir/
- Erbaş, C. (2019). **RIP (Routing Information Protocol) nedir?**.Retrievedfrom https://www.cemerbas.com/2019/06/16/rip-routing-information-protocol/
- Kurose, J. F., & Ross, K. W. (2017). **Computer networking: A top-down approach** (7th ed.). Pearson. Retrieved from https://www.pearson.com/store/p/computer-networking-a-top-down-approach/P100000173324
- Perlman, R. (2000). **Interconnections: Bridges, routers, switches, and internetworking protocols** (2nd ed.). Addison-Wesley. Retrievedf rom https://www.informit.com/store/interconnections-bridges-routers-switches-and-internetworking-9780201634488

- Kurose, J. F., & Ross, K. W. (2013). **Computer Networking: A Top-Down Approach** (6th ed.). Pearson. https://www.pearson.com/store/p/computer-networking-a-top-down-approach/P100000080355
- Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). **Introduction to Algorithms** (3rd ed.). MIT Press. https://mitpress.mit.edu/9780262033848/introduction-to-algorithms/
- Bellman–Ford algorithm. (2023, November 27). In **Wikipedia**. Retrieved from https://en.wikipedia.org/wiki/Bellman%E2%80%93Ford_algorithm
- Dijkstra ve Bellman-Ford En Kısa Yol Algoritmalarının Karşılaştırılması. (2002). **Sakarya Üniversitesi Fen Bilimleri Dergisi**, 6(1), 55-62. Retrieved from https://dergipark.org.tr/tr/pub/saufenbilder/issue/20693/281730
- Optimizing Bellman-Ford Algorithm. (n.d.). Retrieved from https://www.codingdrills.com/tutorial/introduction-to-graph-algorithms/bellmanford-optimization
- Bellman-Ford Algoritması. (2010, May 26). **Bilgisayar Kavramları**. Retrieved from https://bilgisayarkavramlari.com/2010/05/26/bellman-ford-algoritmasi/