BELLMAN-FORD ALGORİTMASI

İLHAN CAN TUĞCU, MUSTAFA DURMAZER

20360859001,21360859041

Bursa Teknik Üniversitesi - Bilgisayar Mühendisliği

İçindekiler

BELLMAN FORD ALGORİTMASI

1.1 Giriş

Algoritma ilk olarak 1955 yılında Alfonso Shimbel tarafından önerildi ancak algoritma ismini 1956 ve 1958'de yayınlanan Richard Bellman ve Lester Ford Jr.'dan aldı. 1959'da Edward F.Moore, algoritmanın farklı bir varyasyonunu yayınlamıştır. Bazı kaynaklarda bu algoritma Bellman-Ford-Moore algoritması olarak da adlandırılır.

1.1.1 Nerelerde Kullanılır

1. Google Maps ve Navigasyon Uygulamaları:

   Örnek:Bir başlangıç noktasından hedefe ulaşmak için alternatif yollar sunar.

   Neden:Bazı yollar negatif ağırlıklar (ör. trafik cezaları, zaman gecikmeleri) içerebilir.

2. Finansal Modeller:

   Örnek:Fırsatları analiz etme.

   Neden:Döviz kuru gibi negatif döngüler içeren durumları tespit edebilir.

3. Sosyal Ağ ve Öneri Sistemleri:

   Örnek:Kullanıcılar arasıındaki minimum mesafeyi bulmak

   Neden:Kullanıcıların benzer ilgi alanlarına göre gruplanmasını sağlar

1.2 Teorik Temel

Bellman-Ford algoritması, bir graf üzerinde negatif ağırlıklı kenarlar içerse bile bir başlangıç düğümünden diğer tüm düğümlere olan en kısa yolları bulur.

Algoritma dijkstra algoritmasında olduğu gibi en küçük değere

sahip olan kenardan gitmek yerine bütün graf üzerindeki kenarları test eder. Bu sayede aç gözlü yaklaşımının handikapına düşmez ve her düğüme sadece bir kere bakarak en kısa yolu bulmuş olur.

1.2.1 Algoritmanın Avantajları

Dijkstra algoritmasının aksine, negatif ağırlıklı kenarlarla çalışabilir.

Grafın negatif ağırlıklı bir döngü içerip içermediğini belirleyebilir.

Dinamik programlama temellerine dayandığından uygulanması kolaydır.

Aç gözlü yaklaşımını (greedy approach) kullanmaz ve her düğüme sadece bir kere bakarak en kısa yolu bulmuş olur.

1.3 Algoritmanın Uygulama Alanları

```
def                                bellman_ford(dugum_sayisi,
kenar_uzunlukları,kaynak_noktası):
    # Graftaki kaynak noktası hariç diğer bütün düğümlerin
ağırlıkları infinitive(sonsuz) olarak belirlenir.
    agırlık = [float('inf')] * dugum_sayisi
    agırlık[kaynak_noktası] = 0
    # Kaynak noktasının ağırlıgı 0 olarak belirlenir.

    for _ in range(dugum_sayisi - 1):
    # dugum_sayisi N ile N-1 tane iterasyon olmalıdır.
        for kaynak, hedef , uzaklık in kenar_uzunlukları:
            if agırlık[kaynak] + uzaklık < agırlık[hedef]:
                agırlık[hedef] = agırlık[kaynak] + uzaklık
                # Her iterasyonda düğümlerin ağırlıkları
yeniden değerlendirilir.




    for kaynak, hedef,uzaklık in kenar_uzunlukları:
        #Eğer (dugum_sayisi-1) tane işlemden sonra hala
ağırlık + uzaklık hedefın agırlıgından küçük çıkıyorsa
negatif döngü vardır
```

```
        if agırlık[kaynak] + uzaklık < agırlık[hedef]:
            print("Negatif ağırlıklı döngü tespit edildi")
            return
            # Negatif ağırlıklı döngü tespit edilirse Bellman-Ford algoritması çalışmaz.Bu yüzden fonksiyon durur.


    for dugum_numarası in range (1,dugum_sayisi):
        print(f"Kaynak Noktasının {dugum_numarası}. düğüme olan uzaklığı: {agırlık[dugum_numarası]}")
        #Eğer negatif ağırlıklı döngü yoksa fonksiyon ağırlıkları yazdırır.


if __name__ == "__main__":
    dugum_sayisi = 6
    # Kenarlar ve aralarındaki mesafeler liste halinde verilmelidir.ÖRN: 0 ile 1. düğüm arasındaki mesafe 6 birimdir.
    kenarlar = [
        (0,1,6),
        (0,2,4),
        (0,3,5),
        (1,4,-1),
        (2,1,-2),
        (2,4,3),
        (3,2,-2),
        (3,5,-1),
        (4,5,3)
    ]
    #Kaynak noktasının hangi nokta olduğu belirtilmelidir.0 olarak belirledik.
    baslangic_dugumu = 0
    bellman_ford(dugum_sayisi,kenarlar,baslangic_dugumu)
    #Kaynak düğümünün diğer düğümlere olan uzaklığı için fonksiyon istenen parametrelerle çağırıldı.
```

```python
def bellman_ford(dugum_sayisi, kenar_uzunlukları,kaynak_noktası):
    # Graftaki kaynak noktası hariç diğer bütün düğümlerin ağırlıkları infinitive(sonsuz) olarak belirlenir.
    agırlık = [float('inf')] * dugum_sayisi
    agırlık[kaynak_noktası] = 0
    # Kaynak noktasının ağırlıgı 0 olarak belirlenir.

    for _ in range(dugum_sayisi - 1):
    # dugum_sayisi N ile N-1 tane iterasyon olmalıdır.
        for kaynak, hedef , uzaklık in kenar_uzunlukları:
            if agırlık[kaynak] + uzaklık < agırlık[hedef]:
                agırlık[hedef] = agırlık[kaynak] + uzaklık
                # Her iterasyonda düğümlerin ağırlıkları yeniden değerlendirilir.



    for kaynak, hedef,uzaklık in kenar_uzunlukları:
        #Eğer (dugum_sayisi-1) tane işlemden sonra hala ağırlık + uzaklık hedefin agırlığından küçük çıkıyorsa negatif döngü vardır
        if agırlık[kaynak] + uzaklık < agırlık[hedef]:
            print("Negatif ağırlıklı döngü tespit edildi")
            return
            # Negatif ağırlıklı döngü tespit edilirse Bellman-Ford algoritması çalışmaz.Bu yüzden fonksiyon durur.


    for dugum_numarası in range (1,dugum_sayisi):
        print(f"Kaynak Noktasının {dugum_numarası}. düğüme olan uzaklığı: {agırlık[dugum_numarası]}")
        #Eğer negatif ağırlıklı döngü yoksa fonksiyon ağırlıkları yazdırır.
```

```python
if __name__ == "__main__":
    dugum_sayisi = 6
    # Kenarlar ve aralarındaki mesafeler liste halinde verilmelidir.ÖRN: 0 ile 1. düğüm arasındaki mesafe 6 birimdir.
    kenarlar = [
        (0,1,6),
        (0,2,4),
        (0,3,5),
        (1,4,-1),
        (2,1,-2),
        (2,4,3),
        (3,2,-2),
        (3,5,-1),
        (4,5,3)
    ]
    #Kaynak noktasının hangi nokta olduğu belirtilmelidir.0 olarak belirledik.
    baslangic_dugumu = 0
    bellman_ford(dugum_sayisi,kenarlar,baslangic_dugumu)
    #Kaynak düğümünün diğer düğümlere olan uzaklığı için fonksiyon istenen parametrelerle çağırıldı.
```

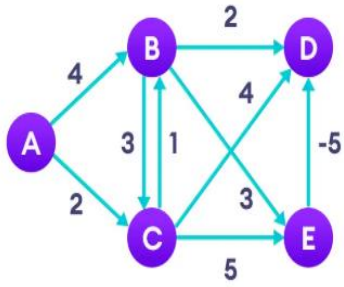
Bellman-Ford Algoritması Finans, Ağ ve İletişim, Ulaşım ve Lojistik, Enerji Sistemleri, Yapay zeka ve Makine Öğrenimi ve Navigasyon sistemlerinde kullanılır.

1.4 Performans Analizi

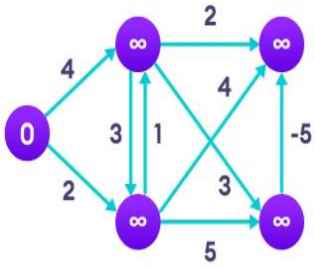
En iyi durum için Bellman Ford Algoritmasının zaman karmaşıklığı O(E) iken, ortalama durum ve en kötü durum zaman karmaşıklığı ise O(N*E)'dir. Burada N düğüm sayısı ve E ağırlığı incelenecek toplam kenar sayısıdır. Ayrıca, Bellman-Ford algoritmasının uzay karmaşıklığı O(N)'dir. Çünkü dizinin boyutu N'dir.

Algoritmayı daha verimli hale getirmek için erken sonlandırma, negatif döngü tespiti, paralel işleme ve yaklaşık hesaplamalar kullanılabilir.

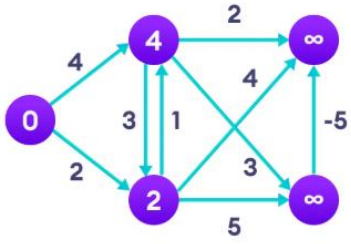
1.5 Şekil, Tablo



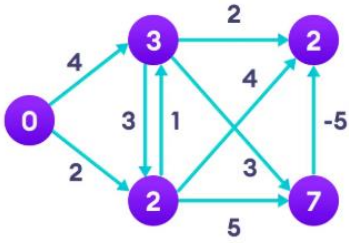
Şekil 1 Başlangıç Durumu



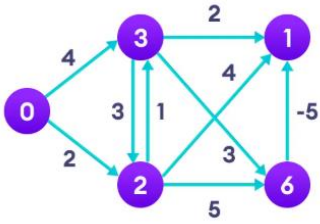
Şekil 1.1 Başlangıç düğümü seçilir ve diğer tüm düğümler sonsuz değer alır.



Şekil 1.2 Her kenar kontrol edilir daha kısa yol varsa güncellenir.



Şekil 1.3 Bu işlem N-1 kez tekrar edilir



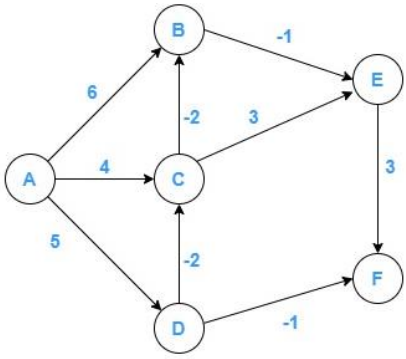
Şekil 1.4 En kısa maliyetli şekilde düğümler tamamlanır.

```python
if __name__ == "__main__":
    dugum_sayisi = 5
    # Kenarlar ve aralarındaki mesafeler liste halinde verilmelidir.ÖRN: 0 ile 1. düğüm arasındaki mesafe 6 birimdir.
    kenarlar = [
        (0,1,4),
        (0,2,2),
        (1,2,3),
        (1,3,2),
        (1,4,3),
        (2,1,1),
        (2,4,5),
        (2,3,4),
        (4,3,-5)
    ]
    #Kaynak noktasının hangi nokta olduğu belirtilmelidir.0 olarak belirledik.
    baslangic_dugumu = 0
    bellman_ford(dugum_sayisi,kenarlar,baslangic_dugumu)
    #Kaynak düğümünün diğer düğümlere olan uzaklığı için fonksiyon istenen parametrelerle çağırıldı.
```

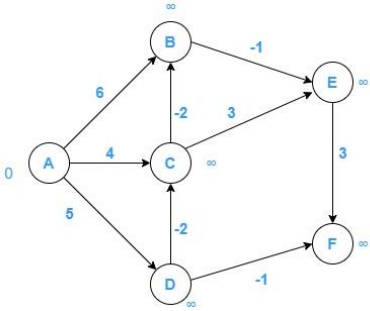
Şekil1.5 Şekil 1'in Koddaki uygulaması

```
Kaynak Noktasının 1. düğüme olan uzaklığı: 3
Kaynak Noktasının 2. düğüme olan uzaklığı: 2
Kaynak Noktasının 3. düğüme olan uzaklığı: 1
Kaynak Noktasının 4. düğüme olan uzaklığı: 6
```

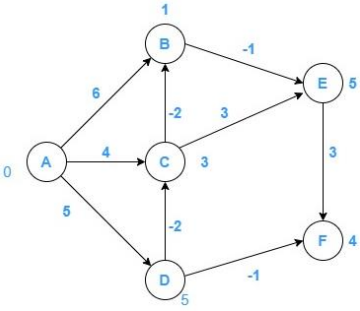
Şekil1.6 Şekil 1'in Koddaki çıktısı



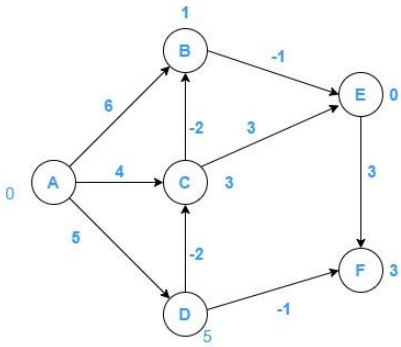
Şekil 2 Başlangıç Grafı



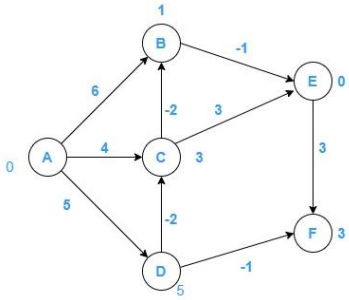
Şekil 2.1 Başlangıç düğümüne 0 atanır, diğer tüm düğümler sonsuz değer alır.



Şekil 2.2 Tüm kenarlar N-1 defa incelenir.



Şekil 2.3 İkinci yinelemeden sonraki durum



Şekil 2.4 Son yinelemeden sonraki durum

A-A: 0

A-B: 1

A-C: 3

A-D: 5

A-E: 0

A-F: 3

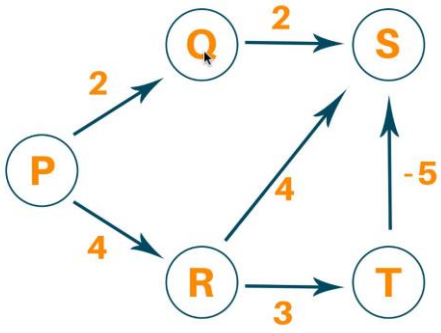
Şekil 2.5 Güncel tablo

```python
if __name__ == "__main__":
    dugum_sayisi = 6
    # Kenarlar ve aralarındaki mesafeler liste halinde verilmelidir.ÖRN: 0 ile 1. düğüm arasındaki mesafe 6 birimdir.
    kenarlar = [
        (0,1,6),
        (0,2,4),
        (0,3,5),
        (1,4,-1),
        (2,1,-2),
        (2,4,3),
        (3,2,-2),
        (3,5,-1),
        (4,5,3)
    ]
    #Kaynak noktasının hangi nokta olduğu belirtilmelidir.0 olarak belirledik.
    baslangic_dugumu = 0
    bellman_ford(dugum_sayisi,kenarlar,baslangic_dugumu)
    #Kaynak düğümünün diğer düğümlere olan uzaklığı için fonksiyon istenen parametrelerle çağırıldı.
```

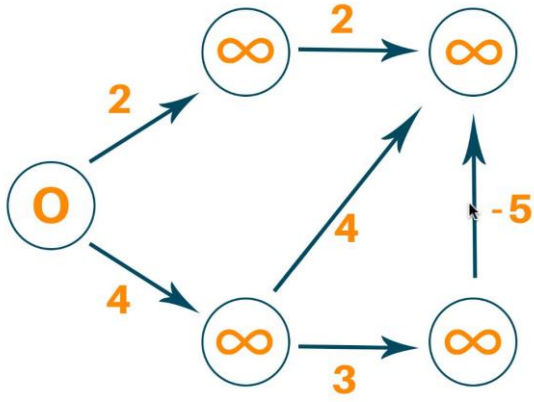
Şekil 2'nin Koda Uygulanması

```
Kaynak Noktasının 1. düğüme olan uzaklığı: 1
Kaynak Noktasının 2. düğüme olan uzaklığı: 3
Kaynak Noktasının 3. düğüme olan uzaklığı: 5
Kaynak Noktasının 4. düğüme olan uzaklığı: 0
Kaynak Noktasının 5. düğüme olan uzaklığı: 3
```

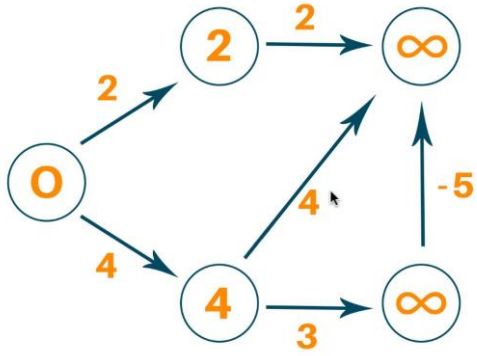
Şekil 2'nin Koddaki Çıktısı



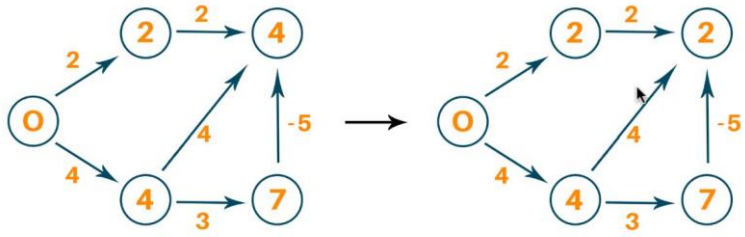
Şekil 3 Başlangıç Grafı



Şekil 3.1 Başlangıç düğümü 0, kalan düğümler sonsuz değer alır.



Şekil 3.2 En kısa maliyetli yollar ile devam edilir.



Şekil 3.3 Sonraki iterasyon

	Q	R	S	T
O	∞	∞	∞	∞
O	2	4	∞	∞
O	2	4	4	7
O	2	4	2	7
O	2	4	2	7

Şekil 3.4 Son durum

```
if __name__ == "__main__":
    dugum_sayisi = 5
    # Kenarlar ve aralarındaki mesafeler liste halinde verilmelidir.ÖRN: 0 ile 1. düğüm arasındaki mesafe 6 birimdir.
    kenarlar = [
        (0,1,2),
        (0,2,4),
        (1,3,2),
        (2,3,4),
        (2,4,3),
        (4,3,-5)
    ]
    #Kaynak noktasının hangi nokta olduğu belirtilmelidir.0 olarak belirledik.
    baslangic_dugumu = 0
    bellman_ford(dugum_sayisi,kenarlar,baslangic_dugumu)
    #Kaynak düğümünün diğer düğümlere olan uzaklığı için fonksiyon istenen parametrelerle çağırıldı.
```

Şekil 3'ün Koda Uygulanması

```
Kaynak Noktasının 1. düğüme olan uzaklığı: 2
Kaynak Noktasının 2. düğüme olan uzaklığı: 4
Kaynak Noktasının 3. düğüme olan uzaklığı: 2
Kaynak Noktasının 4. düğüme olan uzaklığı: 7
```

Şekil 3'ün Koddaki Çıktısı

1.6 Algoritma Özeti

Bellman-Ford algoritması, ağırlıklı yönlü graflarda en kısa yolu bulmak için kullanılan bir algoritmadır ve özellikle negatif ağırlıklı kenarların bulunduğu graflarda, Dijkstra algoritmasının kullanılmadığı durumlarda tercih edilir. Algoritma, başlangıç düğümünden diğer düğümlere olan en kısa yolları bulmak için her bir kenarı N-1 defa tarar ve bu süreçte mesafeleri günceller. Bu yüzden zaman karmaşıklığı

O(N*E)'dir.

Bellman-Ford Algoritmasının önemli bir avantajı, negatif ağırlıklı döngülerin varlığını tespit edebilmesidir. Eğer böyle bir döngü tespit edilirse algoritma mesafeleri sonsuza dek küçülterek günceller. Ancak çalışma süresi O(N*E) olduğu için büyük ve karmaşık graflarda algoritmanın yavaş çalışmasına sebep olabilir. Algoritmayı daha verimli hale getirmek için erken durdurma optimizasyonu gibi teknikler kullanılarak performans arttırılabilir. Negatif ağırlıklı döngüler sebebiyle bazı problemlerin çözümsüz kalabileceği unutulmamalıdır.

1.7 Kaynakça

Shiksha Online. (n.d.). *Introduction to Bellman-Ford algorithm*. Shiksha. Retrieved from https://www.shiksha.com/online-courses/articles/introduction-to-bellman-ford-algorithm/

Upadhyay, S. (n.d.). *Bellman-Ford algorithm*. Simplilearn. Retrieved from https://www.simplilearn.com/tutorials/data-structure-tutorial/bellman-ford-algorithm

Kempepatil, R. (2023). *Advanced study of shortest route problem and its applications—Bellman-Ford algorithm*. ResearchGate. Retrieved from https://www.researchgate.net/publication/371875632_Advanced_study_of_Shortest_Route_Problem_and_its_applications-bellman-_ford_algorithm

Bhadaniya, S. (n.d.). *Bellman-Ford algorithm in Python*. FavTutor. Retrieved from

https://favtutor.com/blogs/bellman-ford-python