T.C.

SAKARYA ÜNİVERSİTESİ

FEN BİLİMLERİ ENSTİTÜSÜ

BİLGİSAYAR VE BİLİŞİM MÜHENDİSLİĞİ ANABİLİM DALI

BİLİŞİM TEKNOLOJİLERİ PR. (YL) (UZAKTAN EĞİTİM)

VERİ YAPILARI VE ALGORİTMALAR 1.ÖDEV

DIJKSTRA ALGORİTMASI

**Hazırlayan**

SEDAT ÖZTÜRK

E235013168

**Öğretim Üyesi**

Prof. Dr. NEJAT YUMUŞAK

MAYIS 2024

Dijkstra algoritması ismini algoritmanın geliştiricisi olan Hollandalı bilgisayar bilimci **Edsger Dijkstra**’dan almaktadır. Dijkstra algoritması, en kısa yol problemi için kullanılan bir graf algoritmasıdır. Bu algoritmanın amacı, çıkış noktası olarak belirlediği noktadan farklı düğümler aracılığıyla en kısa yol ile hedefe ulaşmaktır.

Günümüzde de internet trafiğinin yönlendirilmesinde, oyun programlamada, Google ve Yandex haritaların kullandığı algoritmalardan birisidir. Endüstri Mühendisliği alanında da üretim, lojistik, bir hizmetin en kısa yolu göz önünde bulundurarak kullanıcıya ulaştırmak gibi konular başta olmak üzere birçok optimizasyon probleminin çözümünde faydalanılır.

Dijkstra Algoritması genel olarak, amaçlanan en kısa mesafeye varana kadar doğru mesafenin yaklaşık olarak hesaplanan değerinin daha uygun değerler ile yer değiştirmesi olayını kullanır. Bu olaya “Gevşeme ilkesi” denir.

Algoritmanın Çalışma Mantığı:

* Başlangıç düğümünden diğer düğümlere olan mesafeleri takip eder.
* Başlangıç düğümünden en yakın düğümü seçer ve bu düğümü işaretler.
* İşaretlenen düğümün komşularını kontrol eder ve başlangıç düğümünden bu komşu düğüme olan mesafeyi günceller.
* En kısa mesafeye sahip olmayan düğümleri işaretler ve işlemi tekrarlar.
* Tüm düğümler işaretlendiğinde en kısa yolu elde ederiz.

Algoritma kullanmamız için bazı kuralları vardır. Grafiğimiz ağırlık ve yöne sahip olmalıdır. Kenarların ağırlıkları pozitif değerlerde iken doğru çalışmaktadır. Bunun nedeni, en kısa yol bulunurken kenarların ağırlıklarının eklenmesidir. Yani bir grafta kenarlardan herhangi biri negatif tanımlanırsa algoritma istenilen sonucu bulamayacaktır. Yinede bazı algoritmalar (Bellman-Ford gibi) ile bu problem aşılabilir.

**1. Dizi Veri Yapısı ile Dijkstra Algoritması Örneği:**

Diyelim ki aşağıdaki ağırlıklı grafiğimiz ele alıyoruz:

6

2

7

2

1

5

3

Diyelim ki aşağıdaki ağırlıklı grafiğimiz ele alıyoruz:

6

2

7

2

1

5

3

|  |  |  |  |
| --- | --- | --- | --- |
| **Sıra** | **Düğüm** | **Bağlar** | **Ağırlıklar** |
| 1 | A | B(6), C(7), E(2) | B(6), C(7), E(2) |
| 2 | E | C(5) | C(5), B(6), C(7) |
| 3 | C | B(7), D(10) | B(6), C(7), |
| 4 | B | D(7) |  |
| 5 | D |  |  |

* Başlangıç düğümü **A** olsun.
* Düğümleri bir dizi içinde tutalım: nodes = ['A', 'B', 'C', 'D']
* Başlangıç düğümünden diğer düğümlere olan mesafeleri bir dizi içinde saklayalım: distances = [0, float('inf'), float('inf'), float('inf')]
* Her adımda en yakın düğümü seçmek için bu diziyi tarayalım:
  + İlk adımda **A** işaretlenir ve **C**’ye olan mesafe güncellenir (1).
  + Sonraki adımda **C** işaretlenir ve **D**’ye olan mesafe güncellenir (5).
  + Son olarak **B** işaretlenir ve **D**’ye olan mesafe güncellenir (4).

Sonuç olarak, en kısa yol **A -> C -> D**’dir ve toplam mesafe **5**’tir.

**Zaman Karmaşıklığı Analizi:**

* Dizi veri yapısı: **O(V^2)**
* Hafıza karmaşıklığı: **O(V)**

**2. Bağlı Liste Veri Yapısı ile Dijkstra Algoritması Örneği:**

* Düğümleri bağlı listelerde saklayalım:
* graph = {
* 'A': [('B', 2), ('C', 1)],
* 'B': [('A', 2), ('D', 3)],
* 'C': [('A', 1), ('D', 4)],
* 'D': [('B', 3), ('C', 4)]
* }
* Başlangıç düğümünden diğer düğümlere olan mesafeleri bağlı listelerde tutalım.
* Her adımda en yakın düğümü seçmek için bağlı listeleri tarayalım.

**Zaman Karmaşıklığı Analizi:**

* Bağlı liste veri yapısı: **O((V + E) log V)**
* Hafıza karmaşıklığı: **O(V + E)**

**3. Öncelikli Kuyruk Veri Yapısı ile Dijkstra Algoritması Örneği:**

* Düğümleri öncelikli kuyrukta saklayalım.
* Başlangıç düğümünden diğer düğümlere olan mesafeleri öncelikli kuyrukta tutalım.
* Her adımda en yakın düğümü seçmek için öncelikli kuyruğu kullanalım.

**Zaman Karmaşıklığı Analizi:**

* Öncelikli kuyruk veri yapısı: **O((V + E) log V)**

**Hafıza Karmaşıklığı Analizi:**

* Öncelikli kuyruk veri yapısı: **O(V + E)**

Kısaltmalar

* **O (Büyük O Notasyonu):** Algoritmanın zaman veya hafıza karmaşıklığını ifade eder. Bir algoritmanın çalışma hızını veya hafıza kullanımını analiz etmek için kullanılır. Örneğin, “O(n)” ifadesi, bir algoritmanın çalışma süresinin girdi boyutuna doğrusal olarak bağlı olduğunu ifade eder.
* **V (Düğüm Sayısı):** Graf içindeki düğüm sayısını temsil eder. Örneğin, bir ağaç yapısında V, ağaçtaki düğüm sayısını ifade eder.
* **E (Kenar Sayısı):** Graf içindeki kenar sayısını temsil eder. Örneğin, bir ağaç yapısında E, ağaçtaki kenar sayısını ifade eder.

**Yorum:**

* Dizi veri yapısı hafıza açısından daha az verimlidir ve büyük graf yapısı için uygun değildir.
* Bağlı liste ve öncelikli kuyruk veri yapıları daha etkilidir ve genellikle tercih edilir.
* Öncelikli kuyruk, Dijkstra algoritmasının en iyi performansı gösterdiği veri yapısıdır.