**Algoritma Analizi Ödev**

Ad Soyad: Barash Serbest

Numara: 5220505051

**Graf Teorisi ve Algoritmaları:**

Graph teorisi, matematiksel nesnelerin ve onların arasındaki bağlantıların incelenmesiyle ilgilenen bir matematik dalıdır. Graf, noktalar ve aralarındaki bağlantılar (kenarlar) ile tanımlanan bir nesnedir. Graf teorisi, graf nesnelerinin özelliklerini ve aralarındaki bağlantıları analiz etmek için kullanılan bir daldır. Graf teorisindeki graf nesnesi, matematiksel bir yapıdır ve çeşitli uygulama alanlarında kullanılır. Örnek olarak, ulaşım ağı, sosyal ağ, elektrik ağı ve internet ağı graf nesneleri olarak modellenebilir.

Graf teorisindeki en temel kavramlardan biri, grafın düzenliliğidir. Bir graf düzenli ise, tüm düğümler (noktalar) aynı dereceye (kenar sayısı) sahip olur. Grafın düzenliliği, graf nesnesinin bazı özelliklerini daha kolay analiz etmeyi sağlar.

Graf teorisinde bir başka önemli kavram ise, grafın çizimleri ve temsilleridir. Grafın çizimi, noktaların (düğümlerin) çizilmesi ve aralarındaki bağlantıların (kenarların) gösterilmesiyle oluşur. Grafın temsili, bir grafı matematiksel olarak ifade etmek için kullanılan yöntemdir. Bir grafın temsili, genellikle matris veya liste olarak ifade edilir.

Graf teorisindeki en temel algoritmalar, graf arama algoritmalarıdır. Graf arama algoritmaları, bir graf içinde belirli bir noktadan başlayarak bir hedef noktaya ulaşmak için kullanılır. Graf arama algoritmaları, çeşitli uygulama alanlarında kullanılır. Örneğin, sosyal ağlarda, belirli bir kişiye veya grubu bulmak için graf arama algoritmaları kullanılır.

Graf arama algoritmaları arasında en temel olanları genişlik öncelikli arama (breadth-first search - BFS) ve derinlik öncelikli arama (depth-first search - DFS) algoritmalarıdır. BFS algoritması, herhangi bir düğümden başlayarak grafın her bir düğümüne erişene kadar işlem yapar. DFS algoritması ise, başlangıç düğümünden başlayarak mümkün olan en derine iner ve oradan geri çıkar.

Graf teorisi, özellikle optimize etmek istediğimiz bir sistemin veya problemin modellenmesinde kullanılabilir. Bu durumda, graf teorisi algoritmaları, birçok uygulama alanında kullanılır. Örneğin, yol bulma, ağ yönlendirme, optimizasyon problemleri ve veri işleme gibi alanlarda graf teorisi algoritmaları yaygın olarak kullanılır. Bu algoritmalar, çözümlemesi zor olan problemleri etkili bir şekilde çözmeye yardımcı olur. Bazı popüler graf teorisi algoritmaları arasında Dijkstra, Bellman-Ford, Kruskal ve Prim algoritmaları bulunmaktadır. Bu algoritmalar, farklı problemler için farklı avantajlara sahiptir ve doğru şekilde seçildiğinde, problemi daha hızlı ve daha verimli bir şekilde çözmeye yardımcı olur.

**Dijkstra, Bellman-Ford, Kruskal** ve **Prim** algoritmalarını kısaca tanımlayacak olursak:

**Dijkstra Algoritması:**

Dijkstra algoritması, graf teorisinde kullanılan bir en kısa yol bulma algoritmasıdır. Başlangıç düğümünden hedef düğüme kadar olan en kısa yolu bulmak için kullanılır. Algoritma, minimum maliyeti olan düğüme gitmeyi seçer ve daha sonra bu düğüme bağlı olan düğümleri ziyaret eder. Dijkstra algoritması, pozitif ağırlıklı graf yapısı kullanılarak uygulanır.

**Bellman-Ford Algoritması:**

Bellman-Ford algoritması, graf teorisinde kullanılan bir en kısa yol bulma algoritmasıdır. Dijkstra algoritmasının aksine, negatif ağırlıklı graf yapısı kullanılarak uygulanabilir. Algoritma, her iterasyonda her düğümü en az bir kez ziyaret eder ve minimum maliyeti olan düğümlere doğru ilerler. Bellman-Ford algoritması, negatif döngülerin varlığı durumunda bir hata mesajı üretir.

**Kruskal Algoritması:**

Kruskal algoritması, minimum gerilim ağacı (minimum spanning tree) bulmak için kullanılan bir algoritmadır. Bir grafın minimum gerilim ağacı, grafın tüm düğümlerini içeren bir ağaçtır ve grafın kenarlarındaki toplam ağırlık minimumdur. Kruskal algoritması, grafın tüm kenarlarını ağırlıklarına göre sıralar ve en küçük ağırlıklı kenardan başlayarak ağacı oluşturur.

**Prim Algoritması:**

Prim algoritması, minimum gerilim ağacı bulmak için kullanılan diğer bir algoritmadır. Algoritma, başlangıç düğümünden başlayarak ağaç oluşturur ve her adımda ağacın bir sonraki düğümünü ekleyerek en küçük ağırlıklı kenarı seçer. Kruskal algoritmasından farklı olarak, Prim algoritması her zaman bir ağaç döndürür ve bir grafın tüm kenarlarını ağırlıklarına göre sıralamak gerekmez.

**Breadth First Search (Geniş Öncelikli Arama) Graf Algoritması**

Breadth First Search (BFS) algoritması, bir graf veya ağaç veri yapısında gezinerek, belirli bir özellik veya hedef düğümü bulmak için kullanılan etkili bir arama algoritmasıdır. BFS, ağacın kökünden başlar ve her seviyedeki düğümleri öncelikli olarak ziyaret ederek, daha sonra bir sonraki seviyedeki düğümlere geçer. BFS algoritması, genellikle en kısa yol veya en az maliyetli yolu bulmak gibi birçok uygulamada kullanılır.

BFS algoritması, genellikle Queue adı verilen bir veri yapısı kullanır. Queue, bir veri yapısıdır ve ilk giren ilk çıkar (FIFO) mantığına sahip bir dizi olarak düşünülebilir. BFS algoritması, Queue veri yapısının ilk girdiği düğümü çıkarır ve ziyaret eder. Daha sonra, bu düğümün komşularını bulur ve daha önce ziyaret edilmediyse kuyruğa ekler. Bu işlem, kuyruk boşalana kadar devam eder.

BFS algoritması, genellikle bir graf veya ağaçta belirli bir düğümü veya özelliği aramak için kullanılır. Ayrıca, BFS algoritması genellikle en kısa yolu veya en az maliyetli yolu bulmak için de kullanılır. BFS algoritması, diğer arama algoritmalarına göre daha az maliyetlidir ve bu nedenle genellikle büyük veri yapıları üzerinde kullanılır.

BFS algoritmasının zaman karmaşıklığı O(N) olarak hesaplanır, çünkü her düğümü yalnızca bir kez ziyaret eder. Bu nedenle, BFS algoritması, büyük veri yapılarında bile hızlı ve verimli bir şekilde çalışır. Bununla birlikte, BFS algoritması, bellek kullanımı açısından bazı zorluklar sunabilir. Özellikle, BFS algoritması, tüm düğümleri bir kuyrukta tuttuğu için, bellek kullanımı büyük veri yapıları için sorun olabilir.

Sonuç olarak, BFS algoritması, graf veya ağaç veri yapılarında belirli bir özelliği aramak veya en kısa yolu veya en az maliyetli yolu bulmak için çok kullanışlı bir algoritmadır. BFS algoritması, Queue veri yapısını kullanarak çalışır ve her düğümü yalnızca bir kez ziyaret eder. Bu nedenle, BFS algoritması, hızlı ve verimli bir şekilde çalışır ve büyük veri yapılarında bile etkili bir şekilde kullanılabilir.

**BFS (Breadth First Search) ve DFS (Depth First Search) Graf Algoritmaları Arasındaki Fark**

BFS geniş öncelikli aramayı temsil eder. Buradaki temel amaç iki hedef arasındaki en kısa mesafeyi bulmaktır. BFS’yi anlatırken arada küçük karşılaştırmalar için DFS’yi de bilmek gerekir. DFS’de aramaya başladığı node’dan ulaşabileceği en derin node’a kadar gider, gidecek daha derin bir node kalmadığında geri sarar (backtracking) ve derin node’lara öncelik vererek gezmeye devam eder.

BFS’nin Buna istinaden kullandığı memory boyutu DFS’ye göre fazladır. Burada DFS’nin tersine önceliğimiz derinlik değil de genişlik olacak. BFS kullanırken genelde Queue veri yapısı kullanılır. Queue kullanmakta ki motivasyon FIFO (ilk giren ilk çıkar) mantığının olmasıdır.

BFS’de bir root node belirlenir ve aynı anda tüm node’lar ziyaret edilir. Burada ziyaret edilen bir node’un dolaşılabileceği tüm komşuları queue (kuyruk)’a eklenir.

**BFS (Breadth First Search) Algoritması Tarihçesi**

Breadth First Search (BFS) algoritması, graf veri yapısındaki bir düğümü aramak için kullanılan bir algoritmadır. BFS algoritması, en kısa yol sorununu çözmek için de kullanılan bir arama algoritmasıdır. BFS algoritmasının kökenleri, 19. yüzyıl matematikçileri Leonhard Euler ve Carl Friedrich Gauss tarafından geliştirilen matematiksel teorilere dayanmaktadır.

Ancak, BFS algoritmasının modern formu, 1950'lerde arama algoritmalarına odaklanan bilgisayar bilimcileri tarafından geliştirildi. Bu algoritmalar, bilgisayar bilimcilerinin birçok farklı alanda kullandığı veri yapılarına dayanır. Özellikle, BFS algoritması, genellikle ağaçlar ve grafikler gibi hiyerarşik veya ağ şeklinde yapılar için kullanılır.

BFS algoritması, ilk olarak 1945 yılında matematikçi Konrad Zuse tarafından tanıtıldı. Zuse, BFS algoritmasını, bir elektrik devresindeki arıza noktasını bulmak için kullanmıştır. Ancak, BFS algoritmasının bilgisayar bilimleri alanında kullanımı, 1950'lerde Dijkstra'nın en kısa yol algoritması gibi diğer arama algoritmaları ile yaygınlaşmıştır.

BFS algoritması, 1960'larda ve 1970'lerde bilgisayar bilimciler tarafından geliştirilen ve kullanılan birçok farklı arama algoritmasının temelini oluşturdu. Özellikle, BFS algoritması, 1962'de Herbert Simon, J.C. Shaw ve Allen Newell tarafından geliştirilen bir yapay zekâ programı olan General Problem Solver'da kullanıldı.

BFS algoritması, günümüzde de birçok farklı alanda kullanılmaktadır. Özellikle, BFS algoritması, web sayfaları, sosyal ağlar, haritalar ve çizgi filmler gibi grafiksel veri yapıları için kullanılır. BFS algoritması ayrıca, yapay zekâ, robotik ve genetik algoritmalar gibi farklı bilimsel ve teknolojik alanlarda da kullanılmaktadır.

Sonuç olarak, BFS algoritması, matematiksel teorilerden modern bilgisayar bilimlerine kadar uzanan bir tarihi olan ve günümüzde birçok farklı alanda kullanılan bir arama algoritmasıdır.

**BFS Algoritması Zaman (Time Complexity ) ve Alan Karmaşıklığı (Space Comlexity)**

Breadth First Search (BFS) algoritmasının zaman ve mekân karmaşıklığı, veri yapısının boyutuna ve işlem sayısına bağlıdır.

**Zaman Karmaşıklığı:**

**İyi Durum:** BFS algoritması en kısa yolu bulduğunda, yani aranan düğümün kök düğümden yakınsak bir şekilde bulunduğunda, her düğüm yalnızca bir kez ziyaret edilir. Bu durumda, algoritmanın zaman karmaşıklığı O (1 + E/V) şeklinde ifade edilir. Burada E, kenar sayısı ve V, düğüm sayısıdır. Bu durumda, BFS algoritması O(E/V) zaman karmaşıklığına sahiptir.

**Orta Durum:** Graf yapısı dengeli olmayan bir yapıya sahip olduğunda, BFS algoritması yine her düğümü yalnızca bir kez ziyaret eder. Ancak bu durumda işlem sayısı artar ve zaman karmaşıklığı O(E+V) olur.

**Kötü Durum:** BFS algoritması, aranan düğümün yapının en sonuna kadar gitmesi gerektiğinde, yani kök düğümden uzakta olduğunda, her düğümün tüm komşuları ziyaret edilir. Bu durumda, BFS algoritması O(V+E) zaman karmaşıklığına sahip olur.

**Alan Karmaşıklığı:**

BFS algoritması genellikle Queue adı verilen bir veri yapısı kullanır. Bu nedenle, mekân karmaşıklığı, en fazla eşzamanlı düğüm sayısı kadardır. Yani, BFS algoritmasının mekân karmaşıklığı O(V) olarak ifade edilir.

Özetle, BFS algoritmasının zaman karmaşıklığı, düğümlerin ve kenarların sayısına bağlıdır. Ancak genellikle zaman karmaşıklığı V+E şeklinde ifade edilir. Mekân karmaşıklığı ise yalnızca eşzamanlı düğüm sayısı kadardır, bu da V kadar olabilir.

**BFS (Breadth First Search) Algoritması Akış Şeması**

1. Başlangıç düğümünü belirle ve ziyaret edilmemiş olarak işaretle.

2. Boş bir Queue veri yapısı oluştur ve başlangıç düğümünü bu kuyruğa ekle.

3. Kuyruk boşalana kadar aşağıdaki adımları tekrarla:

a. Kuyruğun en başındaki düğümü çıkar ve ziyaret edilmiş olarak işaretle.

b. Eğer bu düğüm hedef düğümse, algoritmayı durdur ve en kısa yolu bulmuş ol.

c. Düğümün komşularını bul ve ziyaret edilmemiş olanları kuyruğa ekle.

d. Her komşu düğümü için, geçerli düğümün en kısa yol uzaklığını bu komşu düğüme kaydedin.

4. Eğer kuyruk boşaldı ve hedef düğüme ulaşılamadıysa, o zaman hedef düğüme ulaşılamaz.

Yukarıdaki akış şeması, BFS algoritmasını iki nokta arasında en kısa yolu bulmak için kullanmanın temel adımlarını açıklar. Başlangıç düğümü belirlenir ve ziyaret edilmemiş olarak işaretlenir. Daha sonra, başlangıç düğümü Queue veri yapısına eklenir ve kuyruk boşalana kadar aşağıdaki adımlar tekrarlanır.

Kuyruğun en başındaki düğüm çıkarılır ve ziyaret edilmiş olarak işaretlenir. Eğer bu düğüm hedef düğüm ise, algoritma durdurulur ve en kısa yol bulunmuş olur. Daha sonra, düğümün komşuları bulunur ve ziyaret edilmemiş olanları kuyruğa eklenir. Her komşu düğüm için, geçerli düğümün en kısa yol uzaklığı kaydedilir.

Son olarak, eğer kuyruk boşaldı ve hedef düğüme ulaşılamadıysa, o zaman hedef düğüme ulaşılamaz demektir. Bu durumda, BFS algoritması, iki nokta arasındaki en kısa yolu bulamamış demektir.

Bu akış şeması, BFS algoritmasını adım adım takip ederek iki nokta arasındaki en kısa yolu bulmak için kullanabilirsiniz.

**BFS (Breadth First Search) Algoritması Mind Map**

**1.** Başlangıç noktası ve hedef noktası belirlenir.

**2.** BFS algoritması, başlangıç noktasından başlar ve hedef noktasına kadar olan en kısa yolu bulmaya çalışır.

**3.** BFS algoritması, başlangıç noktası ile başlar ve komşu düğümleri keşfeder.

**4.** Komşu düğümler Queue (kuyruk) veri yapısına eklenir.

**5.** Başlangıç düğümünden itibaren ilk seviyedeki tüm düğümler işlenir.

**6.** Daha sonra BFS algoritması, Queue'dan bir sonraki düğümü alır ve onun komşularını keşfetmeye devam eder.

**7.** BFS algoritması, hedef noktaya ulaşıncaya kadar bu işlemi tekrarlar.

**8.** BFS algoritması, hedef düğümü bulduğunda, en kısa yolu bulmak için en düşük maliyetli yolu (yani en az sayıda düğümü) takip eder.

**9.** BFS algoritması, en kısa yolu bulduktan sonra, bu yolu ekrana yazdırır veya işlevsel bir şekilde kullanır.

Böylece, BFS algoritması ile iki nokta arasındaki en kısa yolu bulmak için yukarıdaki adımlar takip edilir.