**7.8 Gözden Geçirme Soruları**

Bu kısa cevaplı, çoktan seçmeli ve doğru/yanlış sorularını yanıtlayarak bölümü ne kadar iyi anladığınızı test edebilirsiniz.

1. Graf tanımında G = (V, E) ifadesindeki V ve E neyi temsil eder?

* Cevap:
  + V (Vertices): Düğümler kümesi (Grafın içinde bulunan noktalar)
  + E (Edges): Kenarlar kümesi (Düğümler arasındaki bağlantılar)

2. Yönlendirilmiş (directed) ve yönlendirilmemiş (undirected) graflarda E’nin tanımındaki fark nedir?

* Cevap:
  + Yönlendirilmiş graf (directed graph): Kenarlar sıralı düğüm çiftleri (u, v) şeklinde tanımlanır ve yönleri vardır. u → v yönünde bir bağlantı vardır ancak v → u olmak zorunda değildir.
  + Yönlendirilmemiş graf (undirected graph): Kenarlar (u, v) veya (v, u) şeklinde iki yönlüdür ve yön farkı gözetilmez. (u, v) kenarı varsa, aynı zamanda (v, u) kenarı da vardır.

3. Derinlik öncelikli aramada (DFS) ziyaret edilen (visited) kümesinin amacı nedir?

* Cevap:
  + Aynı düğümün tekrar ziyaret edilmesini önlemek için kullanılır.
  + Döngülerin (cycles) içinde sonsuz bir döngüye girmeyi engeller.
  + DFS'in daha önce keşfedilmiş düğümleri tekrar kontrol etmesini önleyerek verimli çalışmasını sağlar.

4. Derinlik öncelikli aramada (DFS) geri izleme (backtracking) nasıl gerçekleşir?

* Cevap:
  + DFS, daha önce keşfedilmemiş bir düğüm bulana kadar geriye doğru giderek çalışır.
  + Eğer mevcut düğümün ziyaret edilmemiş komşusu kalmazsa, bir önceki düğüme geri döner.
  + Bu süreç geri izleme (backtracking) olarak adlandırılır ve DFS'in bir dalda ilerleyip çıkmaz sokağa ulaşması durumunda geri dönmesine yardımcı olur.

5. Graf içinde bir yol (path) nedir ve bir döngüden (cycle) farkı nedir?

* Cevap:
  + Yol (path): Bir düğümden başka bir düğüme ulaşmak için takip edilen kenarların sıralı bir dizisidir.
  + Döngü (cycle): Aynı düğümden başlayıp, başka düğümler üzerinden geçerek tekrar aynı düğüme dönen bir yoldur.

6. Ağaç (tree) nedir? Şekil 7.2’deki graf için 0, 1 ve 10 düğümlerini içeren üç ağaç örneği verin.

1. Ağaç 1:

* + Düğümler: {0, 1, 10}
  + Kenarlar: 0 → 10, 1 → 10

1. Ağaç 2:
   * Düğümler: {0, 1, 10, 5, 4}
   * Kenarlar: 0 → 10, 1 → 4, 4 → 5, 5 → 10
2. Ağaç 3:
   * Düğümler: {0, 1, 10, 3, 9}
   * Kenarlar: 3 → 0, 0 → 10, 9 → 10

7. Kruskal algoritması neden minimum ağırlıklı kapsayan ağacı oluştururken hata yapmaz?

* Cevap:
  + Kenarları ağırlıklarına göre sıralayıp en küçük ağırlıklı olanlardan başlayarak ağaca ekler.
  + Bir döngü oluşturmadıkça her zaman en küçük kenarı seçer.
  + Greedy (açgözlü) strateji kullandığı için her adımda doğru karar verir.

8. Dijkstra algoritması neden düğümlere olan yol maliyetlerini hesaplarken hata yapmaz?

* Cevap:
  + Her adımda en düşük maliyetli düğümü seçer.
  + Bir düğüm işaretlendikten sonra maliyeti tekrar güncellenmez, çünkü daha düşük maliyetli bir yol bulunamaz.
  + Dinamik programlama benzeri bir yapı kullandığı için en kısa yolu garantiler.

9. Kruskal algoritması için en iyi graf gösterimi hangisidir ve neden?

* Cevap:
  + Kenar listesi (Edge List) en iyi gösterimdir.
  + Çünkü öncelikle kenarları ağırlıklarına göre sıralaması gerektiğinden, sadece kenar bilgisine ihtiyaç duyar.

10. Dijkstra algoritması neden önceki (previous) düğümü saklar?

* Cevap:
  + Önceki düğüm, en kısa yolu geri takip etmek için gereklidir.
  + Kaynak düğümden herhangi bir düğüme en kısa yolu belirlemek için önceki düğüm bilgisi saklanır.
  + En kısa yolun hangi düğümlerden geçtiğini çıkarmak için kullanılır.

**7.9 Bölüm Problemleri ve Çözümleri**

1. Grafikte 9 ve 29 numaralı düğümler arasındaki yolu bulan bir program yazın.

* Çözüm:
  + Bu problemi çözmek için derinlik öncelikli arama (DFS) veya genişlik öncelikli arama (BFS) kullanabiliriz.
  + DFS ile 9 numaralı düğümden başlayıp 29 numaralı düğüme ulaşana kadar tüm yolları keşfedebiliriz.
  + Program, izlenen yolu yazdırmalıdır.

2. 9 ve 29 düğümleri arasındaki en kısa yolu bulan bir program yazın (Kenarlardaki ağırlıkları dikkate almayın).

* Çözüm:
  + Genişlik Öncelikli Arama (BFS) kullanarak en kısa yolu bulabiliriz.
  + BFS, en kısa yolu bulmak için en uygun algoritmadır çünkü tüm düğümleri katmanı ziyaret eder.

3. Dijkstra Algoritmasını kullanarak 9 numaralı düğümden tüm diğer düğümlere en düşük maliyetle ulaşımı bulan bir program yazın.

* Çözüm:
  + Dijkstra Algoritması, kenar ağırlıkları dikkate alınarak minimum maliyetli yolları bulmak için kullanılır.
  + Python’da heapq modülü kullanarak bir öncelikli kuyruk (priority queue) ile Dijkstra uygulanabilir.

4. Kruskal Algoritmasını kullanarak minimum ağırlıklı kapsama ağacını (MST) bulan bir program yazın.

* Çözüm:
  + Kruskal Algoritması, kenarları ağırlıklarına göre sıralayarak minimum ağırlıklı kapsama ağacı (MST) oluşturur.

5. Verilen grafikte 9 numaralı düğümden 29 numaralı düğüme bir yol bulan bir program yazınız. Ardından kenar ağırlıklarını göz ardı ederek en kısa yolu (kenar sayısı açısından) BFS kullanarak hesaplayınız.

6.Bir grafın iki parçalı (bipartite) olup olmadığını belirleyen bir program yazın.

* Çözüm:
  + İki parçalı graf, tek renkli düğümler arasında kenar bulunmaması gerektiği anlamına gelir.

7.Önceki alıştırmadaki programı genişleterek, grafın iki parçalı (bipartite) olduğu durumda, her iki parçaya ait düğüm kümelerini ekrana yazdıran bir program yazın.

Çözüm:

Bu problem için önce bipartite olup olmadığını kontrol etmemiz gerekiyor. Eğer graf bipartite ise, iki kümeye ayrılmış düğümleri ekrana yazdıracağız. Bunun için aşağıdaki adımları izleyeceğiz:

1. BFS (Genişlik Öncelikli Arama) ile grafı tarayarak düğümleri iki kümeye böleceğiz.
2. Her düğümü farklı renkle (0 veya 1) işaretleyeceğiz.
3. Eğer graf bipartite ise, kümeleri ekrana yazdıracağız.
4. Eğer bipartite değilse, uygun bir mesaj vereceğiz.