Bölüm 15: Soru Cevap

Algoritmalar

Soru

- V düğüm ve E kenarı olan bir çizgenin komşuluk listesi temsili üzerinde BFS (Breadth-First Search) algoritmasının bellek karmaşıklığı nedir?

- A) O(V)
- B) O(E)
- C) O(V + E)
- D) $O(V^2)$

Cevap

- A) O(V)

- BFS, kuyruk (queue) veri yapısı kullanarak düğümleri katman katman ziyaret eder. BFS sırasında:
  - Kuyruk: En fazla V düğüm tutabilir, çünkü her düğüm en fazla bir kez kuyruğa eklenir. Bu, $O(V)$ bellek gerektirir.
  - Ziyaret Edilen Düğümler: Hangi düğümlerin ziyaret edildiğini takip etmek için bir küme veya dizi kullanılır. Bu, $O(V)$ bellek gerektirir.
  - BFS, komşuluk listesine doğrudan erişir ve ek bir bellek gerekmez. Komşuluk listesi $O(V+E)$ yer kaplar, algoritmanın girdisi olarak kabul edilir ve ek bellek hesaplamasına dahil edilmez.

Soru

- Dijkstra algoritmasının zaman karmaşıklığı nedir?

- A) O(E log V)
- B) $O(V^2)$
- C) O(V log V)
- D) O(E + V)

Cevap

- A) O(E log V)

- Dijkstra algoritması, en kısa yol problemi için kullanılır ve zaman karmaşıklığı *O(E log V)* olarak ifade edilir. Burada E, kenar sayısını ve V, düğüm sayısını temsil eder. Dijkstra algoritması, öncelikli kuyruk veri yapısını kullanarak minimum mesafeli düğümleri seçerken her düğümü en fazla bir kez ziyaret eder (O(E)). Öncelik kuyruğunda güncelleme/çıkarma işlemleri O(log V) sürer.

Soru

- Floyd-Warshall algoritmasının, V düğümlü ağırlıklı bir çizge üzerinde tüm çiftler arası en kısa yol bulma işlemi zaman karmaşıklığı nedir?

- A) O(V log V)
- B) $O(V^3)$
- C) O(E log V)
- D) O(E + V)

Cevap

- B) $O(V^3)$

- Floyd-Warshall algoritması, tüm çiftler arası en kısa yol problemini çözmek için kullanılır ve $O(V^3)$ zaman karmaşıklığına sahiptir. Burada V, düğüm sayısını temsil eder. Floyd-Warshall algoritması, dinamik programlama yaklaşımını kullanır ve tüm düğümler arasındaki en kısa yolları bulmak için üçlü iç içe döngüler kullanır.

Soru

- Derinlik-Öncelikli Arama (DFS) algoritmasının, V düğüm ve E kenarı olan bir çizgeyi gezme işlemi için bellek karmaşıklığı nedir?

- A) O(V)
- B) O(E)
- C) O(V + E)
- D) $O(V^2)$

Cevap

- A) O(V)

- Derinlik-Öncelikli Arama (DFS) algoritması, bir çizgeyi gezme işlemi sırasında yığın (stack) veri yapısını kullanır. Her bir düğümü ziyaret ettiğinde, bu düğümün bilgisini yığına ekler ve işlem tamamlandığında yığından çıkarır. Ziyaret edilen düğümleri takip etmek için O(V) ek bellek gerekir. Bu nedenle, DFS algoritmasının bellek karmaşıklığı, en fazla V düğümünü saklamak için gereken bellek miktarıdır.

Soru

- Ağırlıksız çizgenin iki düğümü arasındaki en kısa yolu bulmak için hangi arama algoritması kullanılır?

- A) İkili Arama (Binary Search)
- B) Derinlik-Öncelikli Arama (DFS)
- C) Genişlik-Öncelikli Arama (BFS)
- D) Doğrusal Arama (Linear Search)

Cevap

- C) Genişlik-Öncelikli Arama (BFS)

- Genişlik-Öncelikli Arama (BFS), ağırlıksız bir çizge üzerinde iki düğüm arasındaki en kısa yolu bulmak için kullanılır. BFS, başlangıç düğümünden başlayarak tüm komşu düğümleri keşfeder ve ardından bu düğümlerin komşularını keşfeder. Bu şekilde, BFS her adımda belirli bir uzaklığa (katman) sahip olan düğümleri keşfeder ve en kısa yolu bulur.

Soru

- V düğüm ve E kenardan oluşan bir çizgeyi gezme işlemi için Genişlik-Öncelikli Arama (BFS) algoritmasının zaman karmaşıklığı nedir?

- A) O(V)
- B) O(E)
- C) O(V + E)
- D) $O(V^2)$

Cevap

- C) O(V + E)

- Genişlik-Öncelikli Arama (BFS) algoritması, çizgeyi gezerken kuyruk (queue) veri yapısı kullanır. Her bir düğümü yalnızca bir kez ziyaret ettiği için, BFS algoritmasının zaman karmaşıklığı, her düğümü bir kez işlediği V düğümlerinin sayısına bağlıdır. Ayrıca, her kenarı en fazla bir kez işlediği için, BFS algoritmasının zaman karmaşıklığı aynı zamanda E kenar sayısına da bağlıdır.

Soru

- Rabin-Karp dize eşleştirme algoritmasının en kötü durum zaman karmaşıklığı nedir?

- A) O(n)
- B) O(n + m)
- C) O(nm)
- D) O(n log m)

Cevap

- C) O(nm)

- Rabin-Karp algoritması, metin içinde bir örüntü (pattern) aramak için kullanılan bir dize eşleştirme algoritmasıdır. Ortalama durumda, hash fonksiyonu sayesinde çok hızlı çalışabilir (O(n + m)).
- En kötü durumda, hash çakışmaları (collision) fazla olursa, algoritma her pozisyonda hash eşleşmesinden sonra karakter karakter karşılaştırma yapmak zorunda kalır. Bu durumda, her pozisyonda örüntü uzunluğu kadar karşılaştırma yapılır O(m). Metin uzunluğu n olduğundan toplamda O(nm) karşılaştırma olur.

Soru

- Dinamik programlama kullanarak En Uzun Ortak Alt Dizi (LCS) algoritmasının zaman karmaşıklığı nedir?

- A) O(n)
- B) $O(n^2)$
- C) O(nm)
- D) $O(2^n)$

Cevap

- C) O(nm)

- En Uzun Ortak Alt Dizi (LCS) algoritması, iki dize arasındaki en uzun ortak alt diziyi bulmak için dinamik programlama kullanır. İlk dize uzunluğu *n*, ikinci dize uzunluğu *m* olsun. LCS algoritması, iki dizeyi karşılaştırırken bir matris oluşturur ve bu matrisi doldurur. Matrisin boyutu *n*×*m* olur ve her bir hücreyi doldurmak için sabit zaman gerektirir.

Soru

- Bir metin içinde bir örüntünün tüm tekrarlarını bulmak için kullanışlı olan ve zaman karmaşıklığı O(n + m) olan hangi dize algoritmasıdır?

- A) Rabin-Karp
- B) Knuth-Morris-Pratt (KMP)
- C) Boyer-Moore
- D) Z Algoritması

Cevap

- B) Knuth-Morris-Pratt (KMP)

- Knuth-Morris-Pratt (KMP) algoritması, bir örüntünün (pattern) bir metin içinde tüm eşleşmelerini O(n + m) zamanda bulur.
  - n: metin uzunluğu
  - m: örüntü uzunluğu
- Algoritmanın temel avantajı, daha önce karşılaştırılmış karakterleri tekrar kontrol etmeden ilerlemesidir. Bu da onu örüntünün tüm tekrarlarını verimli şekilde bulmasını sağlar.

Soru

- Yönlü bir çizge içinde güçlü bağlı bileşenleri bulmak için hangi algoritma kullanılır?

- A) Prim algoritması
- B) Floyd-Warshall algoritması
- C) Dijkstra algoritması
- D) Kosaraju algoritması

Cevap

- D) Kosaraju algoritması

- Kosaraju algoritması, yönlü bir çizgenin güçlü bağlı bileşenlerini (strongly connected components) bulmak için kullanılır. İki aşamada çalışır:
  - Çizgenin tamamında DFS uygulanır ve bitiş zamanlarına göre düğümler sıralanır.
  - Çizgenin tersi alınır (tüm okların yönü çevrilir) ve sıralı düğümler üzerinde yeniden DFS ile gezilir.
- Bu işlem sonunda güçlü bağlı bileşenler tespit edilir. Karmaşıklığı O(V+E).

Soru

- Bellman-Ford algoritmasının, negatif kenar ağırlıklarına sahip ağırlıklı çizgede en kısa yolu bulmak için zaman karmaşıklığı nedir?

- A) O(V)
- B) O(E)
- C) O(V log V)
- D) O(VE)

Cevap

- D) O(VE)

- Bellman-Ford algoritması, bir ağırlıklı çizgenin en kısa yolunu bulmak için kullanılır ve negatif kenar ağırlıklarını kabul eder. Algoritma, tüm kenarları V - 1 kez gevşeterek çalışır (burada V düğüm sayısı ve E kenar sayısıdır). Her turda tüm E kenar üzerinde işlem yapılır.

Soru

- Hem yönlü hem yönsüz bir çizge içinde döngü tespit etmek için hangi algoritma kullanılır?

- A) Genişlik-Öncelikli Arama (BFS)
- B) Derinlik-Öncelikli Arama (DFS)
- C) Dijkstra algoritması
- D) Floyd-Warshall algoritması

Cevap

- B) Derinlik-Öncelikli Arama (DFS)

- Derinlik-Öncelikli Arama (DFS) algoritması, bir çizge içinde döngü tespit etmek için kullanılabilir ve hem yönlü hem de yönsüz çizgelerle başa çıkabilir. DFS, çizgenin her düğümünü keşfederken geri izleme yapar ve ziyaret edilen düğümleri bir yığında tutar. Eğer DFS, zaten ziyaret edilmiş bir düğümü tekrar ziyaret ederse, bu durum bir döngü olduğunu gösterir.

Soru

- Kruskal algoritmasının, V düğümü ve E kenarı olan bir çizgenin minimum kapsayan ağacını bulmak için zaman karmaşıklığı nedir?

- A) O(V)
- B) O(E)
- C) O(V log V)
- D) O(E log V)

Cevap

- D) O(E log V)

- Kruskal algoritması, bir çizgenin minimum kapsayan ağacını bulmak için kullanılır. Algoritmanın zaman karmaşıklığı, kenarların ağırlıklarına göre sıralanması ve union-find veri yapısı kullanılarak *O(ElogV)* olur.
- Kruskal, kenarları sıralar (O(E log E)) ve union-find ile kümeleri birleştirir (O(E log V)).

Soru

- Negatif ağırlıklı çizgede, bir düğümden diğerine en kısa yolu bulmak için hangi algoritma kullanılır?

- A. Kruskal
- B. Prim
- C. Dijkstra
- D. Bellman-Ford

Cevap

- Cevap: D

- Bellman-Ford Algoritması, negatif ağırlıklı kenarları içeren çizgeleri de işleyebilen bir en kısa yol algoritmasıdır. Başlangıç düğümünden diğer tüm düğümlere olan en kısa yolları bulur ve negatif ağırlıklı kenar döngülerini de tespit edebilir. Her kenarı belirli bir sayıda (düğüm sayısının bir eksiği kadar) kez güncelleyerek çalışır ve negatif döngüler varsa, bunları algılayıp rapor edebilir.

Soru

- Aşağıdaki çizge problemlerinden hangisi, düğüm ve kenar sayısının toplamına (O(m + n)) göre doğrusal zamanda çözülemez?

- a) Çizgenin bağlı (connected) olup olmadığını belirlemek
- b) Çizgenin iki parçalı (bipartite) olup olmadığını belirlemek
- c) Minimum kapsayan ağacı (MST) belirlemek
- d) Yönlü döngüsüz çizgede düğümlerin topolojik sıralamasını belirlemek

Cevap

- Cevap: C

- Bir çizgenin bağlı veya iki parçalı olup olmadığını belirlemek, derinlik öncelikli arama (DFS) veya genişlik öncelikli arama (BFS) ile yapılabilir. Topolojik sıralama, DFS kullanılarak yapılabilir ve bu algoritma O(m+n) zamanda çalışır. Minimum kapsayan ağaç (MST) problemini çözmek için kullanılan algoritmalar Kruskal ve Prim algoritmalarıdır. Kruskal'ın algoritması kenarları sıralamayı içerir ve bu sıralama O(mlogm) zamanda çalışır. Prim'in algoritması öncelikli kuyruk kullanarak O((m+n)logn) zamanda çalışır.

Soru

- Yönlü çizge G zayıf bağlı (weakly connected) fakat güçlü bağlı (strongly connected) değildir. P = a, b, c, d yolu olduğu biliniyor. Aşağıdakilerden hangisi kesinlikle doğrudur?

- a) (a, b), veya (b, c) ya da (c, d) G'nin bir kenarıdır.
- b) (a, b), (b, c) ve (c, d) G'nin kenarlarıdır.
- c) (b, a), veya (c, b) ya da (d, c) G'nin bir kenarıdır.
- d) (b, a), (c, b) ve (d, c) G'nin kenarlarıdır.

Cevap

- Cevap: A

- Zayıf bağlılık, düğümler arasında yönsüz olarak bir bağlantı olduğunu gösterir, ancak güçlü bağlılık her iki düğüm arasında çift yönlü bağlantıların olduğunu gerektirir. Verilen yol P = a, b, c, d, a'dan d'ye zayıf bir bağlantı sağladığına göre, yönlü grafikte bu düğümler arasında en az bir yönde bir bağlantı olmalıdır.

Soru

- Kruskal algoritmasında, açgözlü (greedy) seçim hangisidir?

- a) Çizgeyi bağlantısız bırakmadan en yüksek maliyetli kenarı çıkarmak.
- b) Bir döngü oluşturmadığı sürece en düşük maliyetli kenarı eklemek.
- c) Mevcut ağaca en az bağlantı maliyetine sahip düğümü eklemek.
- d) Ağaçtan en yüksek bağlantı maliyetine sahip düğümü çıkarmak.

Cevap

- Cevap: B

- Kruskal algoritması, tüm kenarları ağırlıklarına göre artan sırada sıralar. En düşük maliyetli kenardan başlayarak, kenarlar birer birer ağaca (tree) eklenir. Bir kenar eklenirken, döngü oluşturup oluşturmadığı kontrol edilir. Eğer kenarın eklenmesi bir döngü oluşturuyorsa, o kenar eklenmez ve sıradaki en düşük maliyetli kenara geçilir.

Soru

- Huffman algoritmasında, açgözlü (greedy) seçim hangisidir?

- a) Birleştirmek için en düşük değere sahip iki olasılığı seçmek.
- b) Birleştirmek için en yüksek değere sahip iki olasılığı seçmek.
- c) Oluşturulan koda en kısa uzunluktaki kod kelimesini eklemek.
- d) Oluşturulan koddaki en uzun kod kelimesini çıkarmak.

Cevap

- Cevap: A

- Huffman algoritması, tüm sembolleri ve olasılıklarını (frekanslarını) içeren öncelik kuyruğu (min-heap) oluşturur. Her seferinde en düşük olasılığa sahip iki sembolü seçer ve bunları birleştirir. Bu birleştirme işlemi, bu iki sembolü bir ağaç düğümünün alt düğümleri olarak atamayı ve onların olasılıklarının toplamını içeren yeni bir düğüm oluşturmayı içerir. Bu yeni düğüm öncelik kuyruğuna eklenir. Bu işlem, tüm semboller birleşip tek bir ağaç oluşturana kadar tekrarlanır.

Soru

- Dijkstra'nın algoritmasına en çok benzeyen açgözlü algoritma hangisidir?

- a) Kruskal
- b) Prim
- c) Huffman
- d) Floyd-Warshall

Cevap

- Cevap: B

- Prim, Dijkstra'nın algoritmasına benzer. Belirli bir başlangıç noktasından başlayarak, her adımda o anda bilinen en kısa yolu kullanarak ağacı büyütür. Dijkstra gibi, kullandığı veri yapısı öncelik kuyruğudur. Her adımda, ağaçtaki her düğüme olan mesafesini günceller ve bu mesafelere dayanarak en yakın düğümü seçer.

Soru

- Dijkstra ve Prim algoritmaları benzerdir: Her turda, yığından en düşük önceliğe sahip olan u düğümü çıkarılır ve atasıyla bağlantı kurarak oluşturulan ağaca eklenir. (u, v) bir kenar ise, yığında kalan her v düğümü için v'nin önceliği düşürülmeye çalışılır. Eğer başarılı olursa, u v'nin atası yapılır. İki algoritma arasındaki temel farkı nedir?

Cevap

- Her ikisi de yığın (heap) veri yapısını kullanır. İki algoritma arasındaki temel fark, önceliklerin nasıl belirlendiği ve hangi problemleri çözdükleridir.
- Dijkstra Algoritması: Yığında bir düğümün önceliği, kaynak düğümden o düğüme olan en kısa yolun ağırlığıdır. Her adımda, en kısa yolu bulunan düğüm ağaca eklenir.
- Prim Algoritması: Yığında bir düğümün önceliği, o anda oluşturulmuş olan ağaca en yakın olan kenarın ağırlığıdır. Her adımda, ağaca en yakın düğüm ve kenar ağaca eklenir. Ağaçtaki bir düğüme gelen ağırlığı en küçük olan kenar seçilir.

Soru

- Eğer d(x,y) iki sözcük x ve y arasındaki düzenleme mesafesi (edit distance) ise, neden d(x,y) = d(y,x) olduğunu açıklayın.

Cevap

- Her düzenleme işlemi tersine çevrilebilir. Yani, *x*'i *y*'ye dönüştürmek için bir düzenleme işlemi yapılabiliyorsa, aynı işlemler tersine yapıldığında *y*'yi *x*'e dönüştürmek mümkün.
- Örneğin,
  - *x*'e karakter eklemek, *y*'den aynı karakteri silmekle tersine çevrilebilir.
  - *x*'ten karakter silmek, *y*'ye aynı karakteri eklemekle tersine çevrilebilir.
  - Bir karakteri değiştirme işleminin tersi yine bir karakter değişikliğidir.
- Her bir düzenleme işleminin maliyeti (cost) sabittir ve bu maliyet, işlemin tersine çevrilmesi durumunda da aynıdır.

Soru

- n düğüm ve m kenara sahip bir çizgede Kruskal algoritmasının çalışma süresini verin, ve açıklayın.

Cevap

- Kenarları sıralamak, $O(m \log m)$ zaman alır. m, kenar sayısını temsil eder. $m \log m$, $m \log n$'ye eşdeğer veya daha büyüktür çünkü en kötü durumda $m = O(n^2)$ olabilir.
- union-find veri yapısını kullanarak her kenarın iki düğümünün aynı kümeye ait olup olmadığını kontrol etmek ve gerekirse bu kümeleri birleştirmek, amortize edilmiş $O(\alpha(n))$ zaman alır. Burada $\alpha(n)$, ters Ackermann fonksiyonudur ve çok yavaş büyüyen bir fonksiyondur, genellikle pratikte sabit kabul edilir. Bu işlem, m kenar için toplamda $O(m\ \alpha(n))$ zaman alır.
- $O(m \log m) + O(m\ \alpha(n))$

Soru

- Aşağıdaki hangisi ağırlıksız bir çizgede en kısa yolu bulmak için kullanılır?

- A. Derinlik-Öncelikli Arama (DFS)
- B. Genişlik-Öncelikli Arama (BFS)
- C. Dijkstra Algoritması
- D. Prim Algoritması

Cevap

- Cevap: B

- Ağırlıksız çizgede en kısa yolu bulmak için Genişlik-Öncelikli Arama (BFS) kullanılır. BFS, başlangıç düğümünden başlayarak tüm düğümleri seviye seviye (yani genişlik öncelikli) ziyaret eder. Bu şekilde, herhangi bir düğüme ulaşmanın en kısa yolunu garanti eder. Düğümlerin derinliği her seviyede bir artar, bu da başlangıç düğümünden bir düğüme olan yolun uzunluğunu verir.

Soru

- Dijkstra algoritmasının, düğümleri (köşeleri) saklamak için ikili yığın kullanıldığında, zaman karmaşıklığı ne olur?

- A. $O(V^2)$
- B. $O(E + V \log V)$
- C. $O(V^3)$
- D. $O(VE)$

Cevap

- Cevap: B

- İkili yığın, Dijkstra algoritmasında en küçük maliyetli düğümü seçmek için kullanılır. Bu, her çıkarma (extract-min) işleminin O(log V) sürede yapılmasını sağlar. Her kenarın (E) işlenmesi O(1) sürede gerçekleşir. Her düğüm için ekleme (insert) ve güncelleme (decrease-key) işlemleri ikili yığında O(log V) sürede yapılır. Tüm düğümler için toplam süre O(V log V) olur. Dijkstra algoritmasının toplam zaman karmaşıklığı O(E + V log V).

Soru

- Aşağıdakilerden hangisi çizgede negatif ağırlıklı döngüleri tespit edebilir?

- A. Dijkstra
- B. Prim
- C. Bellman-Ford
- D. Floyd-Warshall

Cevap

- Cevap: C

- Bellman-Ford algoritması, başlangıç düğümünden diğer tüm düğümlere olan en kısa yolları bulur. Bu işlem V−1 kez tekrarlanır, burada V düğüm sayısını temsil eder. En kısa yolları bulduktan sonra, algoritma bir ek adım daha yapar. Eğer bu adımda herhangi bir kenar için daha kısa bir yol bulunursa, bu durum çizgede negatif ağırlıklı bir döngü olduğunu gösterir.

Soru

- Kruskal algoritmasının, kenarların zaten sıralı olduğunu varsayarsak, minimum kapsayan ağacı bulma zaman karmaşıklığı nedir?

- A. $O(V^2)$
- B. $O(E\ \alpha(V))$
- C. $O(V \log V)$
- D. $O(VE)$

Cevap

- Cevap: B

- Kenarların sıralı olduğunu varsaydığımızda, bu adımın zaman karmaşıklığı O(1) olur çünkü sıralama yapmaya gerek yoktur. Algoritma, her kenarı işleyerek iki ucu farklı kümede ise birleştirir (union) ve aynı sette ise döngü oluşturmamaya dikkat eder. Birleşim (union) ve bulma (find) işlemlerinin her biri, amortize edilmiş zaman karmaşıklığı α(V) olan yapı kullanılarak yapılabilir (örneğin, disjoint-set veri yapısı). E kenarının her biri için bir birleşim veya bulma işlemi yapılır.

Soru

- Aşağıdakilerden hangisi, yönlü döngüsüz çizgenin (DAG) topolojik sıralaması hakkında doğrudur?

- A. Tekildir (unique).
- B. Döngü olabilir (cyclic).
- C. DFS'in tersten post-order sıralamasını temsil eder.
- D. Doğrusal zamanda gerçekleştirilemez.

Cevap

- Cevap: C

- Topolojik sıralama, bir yönlü döngüsüz çizgede (DAG) düğümleri, her kenar u → v için u düğümü v düğümünden önce gelecek şekilde sıralar. Topolojik sıralama birden fazla olabilir. Topolojik sıralama sadece yönlü döngüsüz çizgeler (DAG) için yapılabilir. Topolojik sıralama, hem DFS ile hem de Kahn algoritması ile O(V + E) zamanda yapılabilir. Yani doğrusal zamanlıdır. Derinlik öncelikli arama (DFS) sırasında her düğüm tamamlandıktan sonra (post-order) bir listeye eklenirse ve bu liste ters çevrilirse, bu liste geçerli bir topolojik sıralamadır.

Soru

- V düğümlü bir çizgenin komşuluk matrisi (adjacency matrix) temsili için alan karmaşıklığı nedir?

- A. O(V)
- B. O(E)
- C. O(V + E)
- D. $O(V^2)$

Cevap

- Cevap: D

- V düğüm sayısına sahip bir çizgenin komşuluk matrisi, V x V boyutunda bir matristir. Her hücre (i, j), i düğümünden j düğümüne bir kenar olup olmadığını belirtir. Eğer kenar varsa 1, yoksa 0 olur. Bu nedenle, toplam hücre sayısı V * V = V^2 olur.

Soru

- Tarjan Algoritması hangi problemi çözmek için kullanılır?

- A. En kısa yol bulma problemi
- B. Minimum kapsayan ağacı bulma problemi
- C. Güçlü bağlantılı bileşenlerin bulunması
- D. Maksimum akış problemi

Cevap

- Cevap: C

- Tarjan Algoritması, yönlü çizgede güçlü bağlantılı bileşenleri (Strongly Connected Components - SCC) bulmak için kullanılır. Algoritma, DFS kullanarak her düğümü ziyaret eder. Her düğüm bir numara ve düşük bağlantı değeri (low-link value) ile işaretlenir. Bu düşük bağlantı değeri, düğümün ulaştığı en düşük numaralı düğümü temsil eder. DFS sırasında düğümleri bir yığında tutar ve SCC'leri tespit etmek için bu yığını kullanır. Bir düğümün düşük bağlantı değeri kendi numarasına eşitse, bu düğüm bir SCC'nin kök düğümüdür ve yığından çıkarılan düğümler bu SCC'yi oluşturur. Doğrusal zamanda O(V + E) çalışır.

Soru

- A* (A Yıldız) algoritmasının birincil kullanımı nedir?

- A. Bir çizgede minimum kapsayan ağacı bulmak
- B. İki düğüm arasındaki en kısa yolu bulmak
- C. Bir çizgede döngüleri tespit etmek
- D. Düğümleri topolojik olarak sıralamak

Cevap

- Cevap: B

- A* algoritması, iki düğüm arasındaki en kısa yolu bulmak için kullanılan bir yol bulma (path finding) algoritmasıdır. A* algoritması, her düğüm için iki değer hesaplar: bir düğüme ulaşmanın gerçek maliyeti ve o düğümden hedefe ulaşmanın tahmini maliyeti (heuristic). Bu değerlerin toplamı, düğümün toplam maliyeti (f) olarak adlandırılır: f(n)=g(n)+h(n).
- Algoritma, ziyaret edilen düğümleri izlemek için iki liste kullanır: açık liste (ziyaret edilmemiş ve değerlendirilecek düğümler) ve kapalı liste (ziyaret edilmiş düğümler). Açık listeden en düşük f maliyetine sahip düğüm seçilir ve komşu düğümler değerlendirilir. Komşu düğümlerin maliyetleri güncellenir ve açık listeye eklenir. Bu adımlar, hedef düğüme ulaşılana kadar tekrarlanır.

Soru

- A* algoritmasının doğru çalışabilmesi için ne gereklidir?

- A. Uygun bir heuristic (öngörücü) fonksiyon
- B. Yalnızca pozitif kenar ağırlıkları
- C. Çizgede döngü olmaması
- D. Tüm düğümler birbirine bağlı olmalı

Cevap

- Cevap: A

- A* algoritmasının doğru ve verimli bir şekilde çalışabilmesi için en önemli gereksinim, uygun bir heuristic (öngörücü) fonksiyondur. Heuristic fonksiyon hedefe olan gerçek en kısa yol maliyetini asla aşmamalıdır. Her düğüm için $h(n) \leq c(n,m) + h(m)$ olmalıdır, burada $c(n,m)$ n ve m düğümleri arasındaki kenar maliyetidir. Bu özellik, algoritmanın verimli çalışmasını ve her düğümün yalnızca bir kez işlenmesini sağlar.

Soru

- Aşağıdakilerden hangisi ağda maksimum akışı artırmak için geçerli bir yöntem değildir?

- A) Orijinal çizgedeki bir kenarın kapasitesini artırmak
- B) Mevcut iki düğüm arasında pozitif kapasiteli yeni bir kenar eklemek
- C) Çizgeden bir kenar kaldırmak
- D) Kaynak ve hedef arasında pozitif kapasiteli yeni bir yol eklemek

Cevap

- Cevap: C

- Bir kenarın kapasitesini artırmak, o kenar üzerinden daha fazla akışın geçmesine izin verir. Bu, ağın toplam maksimum akışını artırabilir. Yeni bir kenar eklemek, ağda yeni yollar oluşturarak toplam akışı artırabilir. Bir kenarın kaldırılması, ağdaki akış yollarını azaltır ve mevcut akışı engelleyebilir.

Soru

- Ford-Fulkerson yönteminde artık (residual) çizgenin amacı nedir?

- A) Artan (augmenting) yolları bulmak
- B) En kısa yolu hesaplamak
- C) Döngüleri tespit etmek
- D) Minimum kapsayan ağaçlarını bulmak

Cevap

- Cevap: A

- Artık çizgenin amacı, ağdaki artıran yolları bulmaktır. Her kenar, orijinal kapasite eksi mevcut akış olarak tanımlanan bir artık kapasiteye sahiptir. Artıran yol, kaynak düğümden hedef düğüme kadar giden ve üzerinde daha fazla akışın gönderilebileceği bir yoldur. Ford-Fulkerson yönteminde, artıran yollar bulunana kadar artık çizge üzerinde yinelemeli olarak arama yapılır. Bu yollar, genişlik öncelikli arama (BFS) veya derinlik öncelikli arama (DFS) kullanılarak bulunur.

Soru

- Dinic'in ağ akışı algoritmasında, her aşamada bloklama akışını bulmak için hangi teknik kullanılır?

- A) Derinlik-Öncelikli Arama (DFS)
- B) Genişlik-Öncelikli Arama (BFS)
- C) Dijkstra Algoritması
- D) Bellman-Ford Algoritması

Cevap

- Cevap: B

- Dinic'in algoritması, maksimum akışı bir dizi aşamada bulur. Her aşamada, bloklama akışı adı verilen bir geçici akış hesaplanır. Bloklama akışı, tüm artıran yolları tıkayan bir akıştır; yani, kaynak düğümden hedef düğüme doğru herhangi bir artıran yol yoktur. Bloklama akışını bulmak için genişlik-öncelikli arama (BFS) kullanılır. İlk adımda, BFS kullanılarak seviyelendirme çizgesi (level graph) oluşturulur. Bu çizge, tüm kenarları ve düğümleri katmanlara ayırır. BFS, her düğümün seviyesini belirler ve yalnızca bu seviyelendirme çizgesindeki yollar kullanılarak artıran yollar aranır.

Soru

- Push-Relabel algoritmasında, "relabel" işleminin amacı nedir?

- A) Bir düğümün yüksekliğini azaltmak
- B) Bir düğümün yüksekliğini artırmak
- C) Yeni bir artıran yol bulmak
- D) Artık kapasiteleri güncellemek

Cevap

- Cevap: B

- Push-Relabel algoritması, her düğüme yükseklik (height) ve fazla akış (excess flow) değeri atar. "Push" işlemi, fazla akışı komşu düğümlere doğru iterken, "relabel" işlemi, tıkanıklık durumunda düğümün yüksekliğini artırarak akışın devam etmesini sağlar. Bir düğümün tüm çıkış kenarları dolu veya hedef düğüm daha yüksekteyse, bu düğümden akış yapılamaz ve bir tıkanıklık meydana gelir. Bu durumda, "relabel" işlemi, tıkanmış düğümün yüksekliğini artırarak akışın devam edebilmesi için yeni yollar açar. Yükseklik artırılırken, düğümün yüksekliği, mevcut komşu düğümlerin yüksekliklerinden en az bir fazlası olacak şekilde güncellenir.

Soru

- İki dizginin en uzun ortak alt dizgisini (longest common substring) doğrusal zamanda bulmak için en uygun algoritma hangisidir?

- A) Dinamik Programlama
- B) Knuth-Morris-Pratt (KMP)
- C) Suffix Tree (Kuyruk Ağacı)
- D) Rabin-Karp

Cevap

- Cevap: C

- İki dizginin en uzun ortak alt dizgisini bulmak için, her iki dizginin de son eklerini içeren tek bir Suffix Tree oluşturulur. Suffix Tree üzerinde, her iki dizgiden de gelen son ekleri içeren en uzun dal (path) bulunarak, en uzun ortak alt dizgi tespit edilir. Bu yöntem, Suffix Tree'nin oluşturulması ve işlenmesi sırasında doğrusal zamanda çalışır (O(n)).

Soru

- İki dizginin en uzun ortak alt dizisini (Longest Common Subsequence, LCS) bulmak için Dinamik Programlama yaklaşımının zaman karmaşıklığı, m ve n uzunlukları için nedir?

- A) O(m + n)
- B) O(mn)
- C) O(m log n)
- D) $O(n^2)$

Cevap

- Cevap: B

- DP, boyutları m x n olan iki boyutlu bir dizi kullanır. Burada m, ilk dizginin uzunluğunu ve n, ikinci dizginin uzunluğunu temsil eder. Bu tabloyu doldurmak için, her karakter çifti (i, j) için bir hücre hesaplanır. Eğer karakterler eşleşiyorsa, bu hücredeki değer bir önceki hücrenin değerine 1 eklenerek hesaplanır. Eşleşmiyorsa, önceki hücrelerden maksimum değer alınır.

Soru

- Bir dizi aktivitenin başlangıç ve bitiş zamanları bilindiğinde, maksimum sayıda çakışmayan aktiviteyi seçmek için kullanılan açgözlü algoritmanın zaman karmaşıklığı nedir?

- A) O(n)
- B) O(n log n)
- C) O(n^2)
- D) O(n $\log^2$ n)

Cevap

- Cevap: B

- Aktiviteleri bitiş zamanlarına göre sıralamak O(n log n) sürer, ardından tek bir tarama ile seçim yapılır (O(n)), toplam O(n log n).

Soru

- 2D düzlemde n noktanın en yakın çiftini bulmak için böl ve fethet yaklaşımıyla çalışan algoritmanın zaman karmaşıklığı nedir?

- A) O(n)
- B) O(n log n)
- C) O(n^2)
- D) O(n $\log^2$ n)

Cevap

- Cevap: B

- Closest Pair algoritması, noktaları x-koordinatına göre sıraladıktan (O(n log n)) sonra böl ve fethet ile O(n log n) sürede çalışır.

Soru

- Bir dizi kelimeyi saklamak ve aramak için Trie veri yapısının ekleme işlemi için zaman karmaşıklığı nedir (w kelime uzunluğu, n kelime sayısı)?

- A) O(w)
- B) O(nw)
- C) O(w log n)
- D) O(n log w)

Cevap

- Cevap: A

- Trie’de bir kelimeyi eklemek, kelimenin uzunluğu (w) kadar işlem gerektirir, toplam O(w).

SON