GİRESUN ÜNİVERSİTESİ MÜHENDİSLİK FAKÜLTESİ
BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ
ALGORİTMALAR DERSİ BÜTÜNLEME SINAVI

Adı – Soyadı – Numarası:

Soru 1: Aşağıda genişlik öncelikli arama (BFS) algoritmasının kodu verilmiştir.

```
void bfs(int baslangic, List<List<Integer>> komsuluk, int dugumSayisi) {
	Queue<Integer> kuyruk = new LinkedList<>();
	boolean[] ziyaretEdildi = new boolean[dugumSayisi];
	ziyaretEdildi[baslangic] = true;
	kuyruk.add(baslangic);
	while (!kuyruk.isEmpty()) {
		int u = kuyruk.poll();
		System.out.print(u + " ");
		for (int v : komsuluk.get(u)) {
			if (!ziyaretEdildi[v]) {
				ziyaretEdildi[v] = true;
				kuyruk.add(v);
			}
		}
	}
}
```

(a) Kodu inceleyerek algoritma karmaşıklığını bulunuz ve açıklayınız.
(b) Komşuluk listesi yerine komşuluk matrisi kullanılsaydı algoritma bellek ve zaman karmaşıklığı nasıl etkilenirdi açıklayınız.
(c) Kuyruk veri yapısı yerine dizi kullanılsaydı algoritma bellek ve zaman karmaşıklığı nasıl etkilenirdi açıklayınız.

(a)

Zaman Karmaşıklığı:

Genişlik Öncelikli Arama (BFS), bir çizgedeki tüm düğümleri ve kenarları bir kez ziyaret eder.

Kuyruk işlemleri: Her düğüm en fazla bir kez kuyruğa eklenir ve çıkarılır. Bu, düğüm sayısı V kadar işlem gerektirir. Kuyruk işlemleri (ekleme ve çıkarma) O(1) zaman alır. Dolayısıyla, toplamda O(V).

Komşu tarama: Her düğüm için komşuluk listesindeki tüm komşular taranır. Bir grafikte toplam kenar sayısı E ise, tüm komşuların taranması O(E) zaman alır (her kenar bir kez incelenir).

Toplam zaman karmaşıklığı: O(V + E). Bu, düğüm ve kenar sayısına bağlı olarak doğrusal bir karmaşıklıktır.

Bellek Karmaşıklığı:

Kuyruk: En kötü durumda, kuyruk tüm düğümleri tutabilir (örneğin, bir düğümün tüm komşuları kuyruğa eklenirse). Bu, O(V) bellek gerektirir.

Ziyaret dizisi: Her düğüm için bir boolean değer tutulur, bu da O(V) bellek kullanır.

Komşuluk listesi: Komşuluk listesi O(V + E) bellek kullanır (her düğüm için bir liste ve her kenar için bir girdi).

Toplam bellek karmaşıklığı: O(V + E).

Başarılar.. Dr. Öğr. Üyesi Sercan KÜLCÜ 13.06.2025

(b)

Zaman Karmaşıklığı:

Komşuluk matrisi, V x V boyutunda bir matristir. Her düğümün komşularını bulmak için, o düğümün satırındaki tüm değerler kontrol edilmelidir (kenar var mı yok mu diye).

Her düğüm için O(V) zaman harcanır ve V düğüm olduğu için, komşu tarama işlemi O(V^2) zaman alır.

Kuyruk işlemleri O(V) zaman alır.

Toplam zaman karmaşıklığı: O(V^2). Bu, komşuluk listesine kıyasla kötüdür, özellikle seyrek çizgeler için.

Bellek Karmaşıklığı:

Komşuluk matrisi: V x V boyutunda olduğu için O(V^2) bellek gerektirir. Bu, komşuluk listesinin O(V + E) belleğine kıyasla fazladır, özellikle seyrek çizgeler için.

Kuyruk ve ziyaret dizisi: O(V) bellek kullanır.

Toplam bellek karmaşıklığı: O(V^2), çünkü komşuluk matrisinin etkisi vardır.

Açıklama: Komşuluk matrisi, yoğun çizgeler için uygun olabilir, ancak seyrek çizgeler için hem zaman hem de bellek açısından verimsizdir.

(c)

Zaman Karmaşıklığı:

Kuyrukta, eleman ekleme ve çıkarma işlemleri O(1) zaman alır (LinkedList kullanıldığında).

Bir dizi kullanılsaydı:

Ekleme: Dizinin sonuna eleman eklemek O(1) (amortize edilmiş, dinamik dizi kullanıldığında). Ancak, dizi boyutunun artırılması gerektiğinde O(n) zaman alabilir.

Çıkarma: Diziden ilk elemanı çıkarmak için tüm elemanların kaydırılması gerekir, bu da O(n) zaman alır (n, dizideki eleman sayısı). Alternatif olarak, bir işaretçi kullanılarak ilk eleman "çıkarılabilir", ancak bu da ek yönetim gerektirir.

Komşu tarama: Dizi, komşu tarama işlemini değiştirmez, bu yüzden O(E) kalır.

Toplam zaman karmaşıklığı: Çıkarma işlemlerinin O(n) olması nedeniyle, her düğüm için çıkarma yapıldığında toplam O(V^2) zaman alabilir (en kötü durumda). Bu, kuyruk kullanımına kıyasla kötüdür.

Bellek Karmaşıklığı:

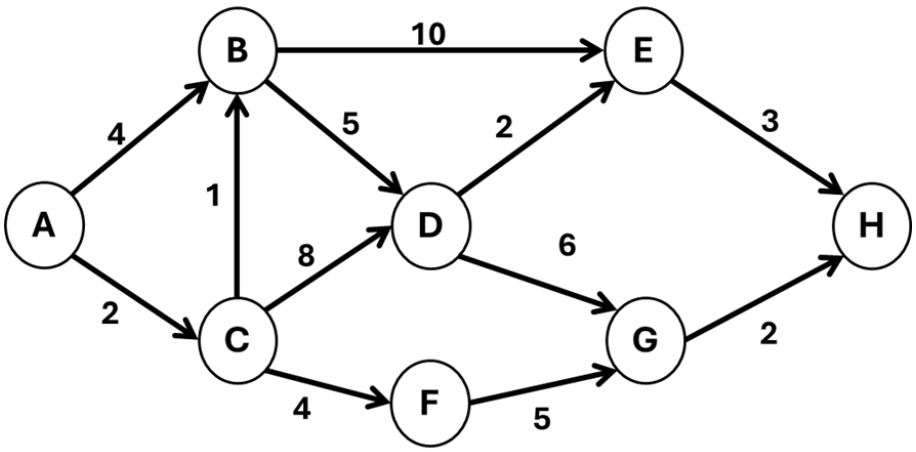
Dizi: Kuyruk gibi, en kötü durumda O(V) düğüm tutar, bu yüzden bellek kullanımı benzerdir.

Ziyaret dizisi ve komşuluk listesi: Değişmez, yani O(V + E).

Toplam bellek karmaşıklığı: O(V + E), kuyrukla aynıdır.

Açıklama: Dizi, kuyruğun sağladığı sabit zamanlı ekleme ve çıkarma avantajını ortadan kaldırır. Özellikle çıkarma işlemi, zaman karmaşıklığını artırır. Kuyruk, BFS için optimize bir veri yapısıdır ve dizi kullanımı algoritmanın performansını düşürür.

Soru 2: Aşağıda 8 düğüm ve 12 kenardan oluşan yönlü ve ağırlıklı çizge verilmiştir. Soruları açıklayarak cevaplayınız.



(a) A düğümünden başlayarak BFS algoritmasına göre gezildiğinde oluşan çıktıyı yazınız.
(b) A düğümünden başlayarak DFS algoritmasına göre gezildiğinde oluşan çıktıyı yazınız.
(c) A düğümünden diğer düğümlere olan uzaklığı Dijkstra en kısa yol algoritması ile bulunuz.
(d) Prim algoritmasını kullanarak çizgenin minimum kapsayan ağacını (MST) bulunuz.

Çizge Tanımı

Düğümler: A, B, C, D, E, F, G, H

Kenarlar (yönlü ve ağırlıklı):

A → B (ağırlık: 4) A → C (ağırlık: 2) B → D (ağırlık: 5) B → E (ağırlık: 10)

C → B (ağırlık: 1) C → D (ağırlık: 8) C → F (ağırlık: 4) D → E (ağırlık: 2)

D → G (ağırlık: 6) E → H (ağırlık: 3) F → G (ağırlık: 5) G → H (ağırlık: 2)

Not: Çizge yönlü olduğu için kenarlar tek yönlüdür (örneğin, A → B var, ama B → A yok). A kaynak, H hedef, diğerleri ara düğümdür. Bu çizge, 12 kenar içerir ve tüm düğümler birbirine dolaylı olarak bağlıdır.

(a) BFS Algoritmasına Göre Gezinti

BFS (Genişlik Öncelikli Arama), düğümleri katman katman gezer ve bir kuyruk kullanır. Yönlü çizgede, sadece gidilebilen yönler dikkate alınır. A'dan başlıyoruz.

Adımlar:

1. Başlangıç: Kuyruk = [A], ziyaret edildi = {A}
2. A çıkar: A'nın komşuları B ve C. Kuyruk = [B, C], ziyaret edildi = {A, B, C}
3. B çıkar: B'nin komşuları D ve E. Kuyruk = [C, D, E], ziyaret edildi = {A, B, C, D, E}
4. C çıkar: C'nin komşuları B, D (B, D, F). B ve D zaten ziyaret edildi, sadece F eklenir. Kuyruk = [D, E, F], ziyaret edildi = {A, B, C, D, E, F}
5. D çıkar: D'nin komşuları E ve G. E zaten ziyaret edildi, G eklenir. Kuyruk = [E, F, G], ziyaret edildi = {A, B, C, D, E, F, G}
6. E çıkar: E'nin komşusu H. Kuyruk = [F, G, H], ziyaret edildi = {A, B, C, D, E, F, G, H}
7. F çıkar: F'nin komşusu G (ziyaret edildi). Kuyruk = [G, H]
8. G çıkar: G'nin komşusu H (ziyaret edildi). Kuyruk = [H]
9. H çıkar: H'nin komşusu yok. Kuyruk boş, algoritma biter.

Çıktı: A B C D E F G H

(b) DFS Algoritmasına Göre Gezinti

DFS (Derinlik Öncelikli Arama), bir dalı sonuna kadar gezer, ardından geri döner. Yönlü çizgede, yine sadece gidilebilen yönler dikkate alınır. A'dan başlıyoruz. Varsayılan olarak, komşular alfabetik sırayla seçilir (örneğin, A'nın komşuları B ve C ise önce B seçilir).

Adımlar:

1. A'yı ziyaret et: A'nın komşuları B, C.
2. B'yi ziyaret et: B'nin komşuları D, E.
3. D'yi ziyaret et: D'nin komşuları E, G.
4. E'yi ziyaret et: E'nin komşusu H.
5. H'yi ziyaret et: H'nin komşusu yok, geri dön.
6. G'yi ziyaret et: G'nin komşusu H (ziyaret edildi), geri dön.
7. E zaten ziyaret edildi, geri dön.
8. C'yi ziyaret et: C'nin komşuları B, D, F.
9. B zaten ziyaret edildi, geç.
10. D zaten ziyaret edildi, geç.
11. F'yi ziyaret et: F'nin komşusu G (ziyaret edildi), geri dön.

Çıktı: A B D E H G C F

Başarılar.. Dr. Öğr. Üyesi Sercan KÜLCÜ 13.06.2025

(c) Dijkstra En Kısa Yol Algoritması ile Uzaklıklar

Dijkstra algoritması, A'dan diğer tüm düğümlere en kısa yolları bulur. Yönlü ve ağırlıklı çizge için ağırlıkları kullanıyoruz.

Başlangıç:

- Uzaklıklar: d[A] = 0, diğerleri ∞
- Ziyaret edilmemiş düğümler: {A, B, C, D, E, F, G, H}
- Öncelik kuyruğu: [(A, 0)]

Adımlar:

1. A'yı seç (d[A] = 0): Komşular B(4), C(2). Güncelle: d[B] = 4, d[C] = 2. Kuyruk: [(C, 2), (B, 4)]
2. C'yi seç (d[C] = 2): Komşular B(2+1=3), D(2+8=10), F(2+4=6). Güncelle: d[B] = 3, d[D] = 10, d[F] = 6. Kuyruk: [(B, 3), (F, 6), (D, 10)]
3. B'yi seç (d[B] = 3): Komşular D(3+5=8), E(3+10=13). Güncelle: d[D] = 8, d[E] = 13. Kuyruk: [(F, 6), (D, 8), (E, 13)]
4. F'yi seç (d[F] = 6): Komşu G(6+5=11). Güncelle: d[G] = 11. Kuyruk: [(D, 8), (E, 13), (G, 11)]
5. D'yi seç (d[D] = 8): Komşular E(8+2=10), G(8+6=14). Güncelle: d[E] = 10, d[G] = 11 (değişmez). Kuyruk: [(E, 10), (G, 11)]
6. E'yi seç (d[E] = 10): Komşu H(10+3=13). Güncelle: d[H] = 13. Kuyruk: [(G, 11), (H, 13)]
7. G'yi seç (d[G] = 11): Komşu H(11+2=13). Güncelle: d[H] = 13 (değişmez). Kuyruk: [(H, 13)]
8. H'yi seç (d[H] = 13): Komşu yok. Algoritma biter.

En Kısa Uzaklıklar:

- d[A] = 0
- d[B] = 3 (A → C → B)
- d[C] = 2 (A → C)
- d[D] = 8 (A → C → B → D)
- d[E] = 10 (A → C → B → D → E)
- d[F] = 6 (A → C → F)
- d[G] = 11 (A → C → F → G)
- d[H] = 13 (A → C → B → D → E → H veya A → C → F → G → H)

Başarılar.. Dr. Öğr. Üyesi Sercan KÜLCÜ 13.06.2025

(d) Prim Algoritması ile Minimum Kapsayan Ağaç (MST)

Prim algoritması, minimum kapsayan ağacı bulur, ancak çizge yönlü olduğu için MST'yi bulmak için yönleri göz ardı edip **yönsüz** bir çizge olarak ele alınır. Yönsüz çizgede, her kenar çift yönlü kabul edilir ve aynı ağırlık kullanılır.

Yönsüz Çizge Kenarları:

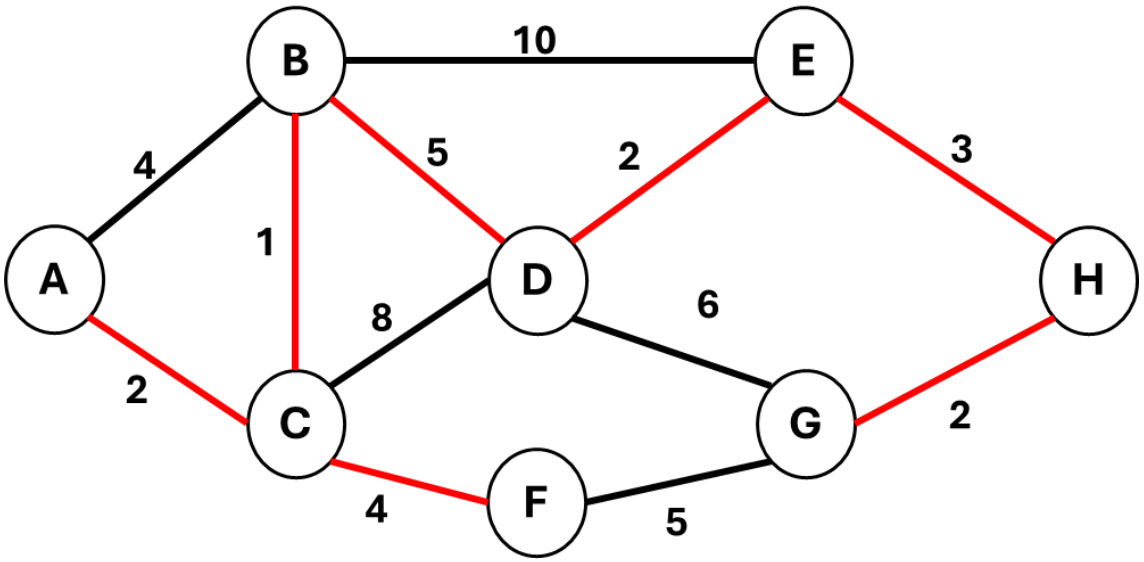
- A ↔ B (4), A ↔ C (2), B ↔ C (1), B ↔ D (5), B ↔ E (10), C ↔ D (8), C ↔ F (4), D ↔ E (2), D ↔ G (6), E ↔ H (3), F ↔ G (5), G ↔ H (2)

Prim Algoritması:

- Başlangıç: A'dan başla, ziyaret edildi = {A}, MST = {}
- Öncelik kuyruğu: [(A-B, 4), (A-C, 2)]

1. En küçük kenar A-C (2): C'yi ekle, MST = {A-C}. Kuyruk: [(A-B, 4), (C-B, 1), (C-D, 8), (C-F, 4)]
2. En küçük kenar C-B (1): B'yi ekle, MST = {A-C, C-B}. Kuyruk: [(A-B, 4), (C-F, 4), (B-D, 5), (B-E, 10), (C-D, 8)]
3. A-B zaten B'yi içeriyor, atla. C-F (4): F'yi ekle, MST = {A-C, C-B, C-F}. Kuyruk: [(B-D, 5), (F-G, 5), (C-D, 8), (B-E, 10)]
4. B-D (5): D'yi ekle, MST = {A-C, C-B, C-F, B-D}. Kuyruk: [(F-G, 5), (D-G, 6), (C-D, 8), (B-E, 10), (D-E, 2)]
5. D-E (2): E'yi ekle, MST = {A-C, C-B, C-F, B-D, D-E}. Kuyruk: [(F-G, 5), (D-G, 6), (C-D, 8), (B-E, 10), (E-H, 3)]
6. E-H (3): H'yi ekle, MST = {A-C, C-B, C-F, B-D, D-E, E-H}. Kuyruk: [(F-G, 5), (D-G, 6), (C-D, 8), (B-E, 10), (G-H, 2)]
7. G-H (2): G'yi ekle, MST = {A-C, C-B, C-F, B-D, D-E, E-H, G-H}. Kuyruk: [(F-G, 5), (D-G, 6), (C-D, 8), (B-E, 10)]
8. Tüm düğümler eklendi, algoritma biter.

GİRESUN ÜNİVERSİTESİ MÜHENDİSLİK FAKÜLTESİ
BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ
ALGORİTMALAR DERSİ BÜTÜNLEME SINAVI



Minimum Kapsayan Ağaç:

- Kenarlar: A-C (2), C-B (1), C-F (4), B-D (5), D-E (2), E-H (3), G-H (2)
- Toplam ağırlık: 2 + 1 + 4 + 5 + 2 + 3 + 2 = 19

Başarılar.. Dr. Öğr. Üyesi Sercan KÜLCÜ 13.06.2025