

Elbette, veri yapıları sınavı için çeşitli konular ve örnek sorular hazırlayabilirim. Veri yapıları, bilgisayar bilimlerinin temel konularından biridir ve genellikle şu başlıkları kapsar:

1. **Temel Veri Yapıları**
   * Diziler
   * Bağlı Listeler (Singly Linked List, Doubly Linked List, Circular Linked List)
   * Yığınlar (Stacks)
   * Kuyruklar (Queues) ve Çift uçlu kuyruklar (Deques)
2. **Ağaçlar**
   * İkili Ağaçlar (Binary Trees)
   * İkili Arama Ağaçları (Binary Search Trees - BST)
   * Denge Ağaçları (AVL Trees, Red-Black Trees)
   * Heap Ağaçları (Binary Heaps, Min-Heaps, Max-Heaps)
   * B-Ağaçları (B-Trees)
3. **Grafik**
   * Temel Grafik Teorisi (Vertices, Edges, Paths, Cycles)
   * Grafik Temsilleri (Adjacency Matrix, Adjacency List)
   * Derinlemesine Arama (Depth-First Search - DFS)
   * Genişlemesine Arama (Breadth-First Search - BFS)
   * Minimum Örtücü Ağaçlar (Minimum Spanning Trees - Kruskal, Prim)
   * En Kısa Yol Algoritmaları (Dijkstra, Bellman-Ford)
4. **Hashing**
   * Hash Tabloları
   * Çarpışma Çözme Teknikleri (Chaining, Open Addressing)
5. **Sıralama ve Arama Algoritmaları**
   * Sıralama (Bubble Sort, Selection Sort, Insertion Sort, Merge Sort, Quick Sort, Heap Sort)
   * Arama (Binary Search, Linear Search)

Örnek sınav soruları:

**1. Temel Veri Yapıları Sorusu**

**Soru:** Bağlı bir listede bir düğümü silmek için kullanılan genel algoritmayı yazın ve zaman karmaşıklığını açıklayın.

**Cevap:**

markdown

Copy code

Algoritma: 1. Silinecek düğüm bulunur. 2. Silinecek düğümün bir önceki düğümünün next işaretçisi, silinecek düğümün next işaretçisine ayarlanır. 3. Silinecek düğümün belleği serbest bırakılır. Zaman Karmaşıklığı: - En iyi durum: O(1) (düğüm başta veya sonda ise) - En kötü durum: O(n) (düğüm listenin sonunda veya bulunması gereken yer listenin sonunda ise)

**2. Ağaçlar Sorusu**

**Soru:** Bir ikili arama ağacının (BST) denge faktörünü açıklayın ve AVL ağacının nasıl çalıştığını özetleyin.

**Cevap:**

markdown

Copy code

Denge Faktörü: - Denge faktörü, bir düğümün sol alt ağacının yüksekliği ile sağ alt ağacının yüksekliği arasındaki farktır. - AVL ağacında, her düğümün denge faktörü -1, 0 veya 1 olabilir. AVL Ağacı: - İkili arama ağacının özel bir türüdür. - Her ekleme veya silme işleminden sonra, ağaç dengede kalacak şekilde rotasyonlar (tekli veya ikili) kullanılarak yeniden düzenlenir. - Bu sayede AVL ağacı her zaman O(log n) zaman karmaşıklığına sahip olur.

**3. Grafik Sorusu**

**Soru:** Depth-First Search (DFS) algoritmasını açıklayın ve bir örnek üzerinde çalıştırın.

**Cevap:**

markdown

Copy code

Algoritma: 1. Bir başlangıç düğümü seçin. 2. Ziyaret edilmemiş komşu düğümü seçin ve DFS'yi o düğüm üzerinde tekrar çalıştırın. 3. Tüm komşular ziyaret edilene kadar 2. adımı tekrarlayın. 4. Geriye doğru dönerek diğer düğümleri ziyaret edin. Örnek:

DFS Sırası: A, B, E, F, C, D, G

bash

Copy code

### 4. Hashing Sorusu \*\*Soru:\*\* Bir hash tablosunda çarpışma çözmek için kullanılan chaining (zincirleme) yöntemini açıklayın. \*\*Cevap:\*\* ```markdown Chaining Yöntemi: - Her hash tablosu slotu için bir bağlantılı liste tutulur. - Hash fonksiyonu aynı indeksi veren anahtarlar bu bağlantılı listede saklanır. - Anahtarlar aynı indekse hash edilirse, bu anahtarlar listeye eklenir. - Bu şekilde çarpışmalar çözülür ve hash tablosu verimli bir şekilde kullanılır.

Bu soruların her biri, farklı veri yapıları konularını kapsamaktadır. Öğrencilerin bu soruları cevaplayarak konuların detaylarını anlamaları ve algoritmaları pratik etmeleri beklenir.

Kuyruk veri yapısı, doğrusal olmayan veri yapıları için yardımcı bir veri yapısı olarak kullanılamaz. false true

ChatGPT

"Kuyruk veri yapısı, doğrusal olmayan veri yapıları için yardımcı bir veri yapısı olarak kullanılamaz." ifadesi yanlıştır (false).

Kuyruk veri yapısı, doğrusal olmayan veri yapıları için sıkça kullanılan yardımcı bir veri yapısıdır. Örneğin:

1. **Ağaç Taraması:** Genişlik öncelikli arama (BFS) algoritmasında, ağaç veya grafiği katman katman taramak için kuyruk veri yapısı kullanılır. BFS, bir başlangıç düğümünden başlayarak, tüm komşu düğümleri bir kuyrukta sıraya koyar ve sırayla işler.
2. **Grafik Algoritmaları:** Yine BFS algoritmasında, grafiklerin bağlantılı bileşenlerini keşfetmek, en kısa yol problemlerini çözmek veya minimum örtücü ağaçları bulmak için kuyruklar kullanılır.
3. **Yazıcı İş Kuyruğu:** Bir yazıcının işlerini sıraya koymak için kuyruk kullanılır, bu da sistemdeki diğer doğrusal olmayan yapıların verimli bir şekilde çalışmasını sağlar.

Bu nedenle, kuyruklar doğrusal olmayan veri yapıları için de önemli bir yardımcı araçtır.

Özetle, "Kuyruk veri yapısı, doğrusal olmayan veri yapıları için yardımcı bir veri yapısı olarak kullanılamaz." ifadesi **yanlıştır**.

Diziler düzenli ya da düzensiz olarak tanımlanabilir. false true

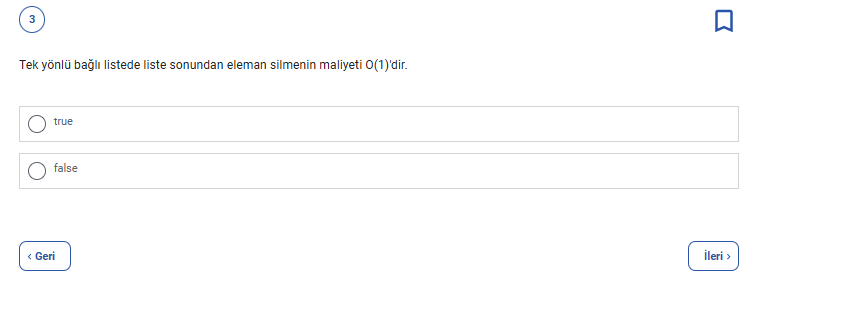
ChatGPT

"Diziler düzenli ya da düzensiz olarak tanımlanabilir." ifadesi **doğrudur** (true).

Diziler, elemanlarının belirli bir sıraya göre düzenlenip düzenlenmediğine göre iki farklı kategoriye ayrılabilir:

1. **Düzenli Diziler (Sorted Arrays):**
   * Elemanları belirli bir düzene göre sıralanmış dizilerdir.
   * Örneğin, artan ya da azalan sıraya göre sıralanmış olabilirler.
   * Sıralı dizilerde arama, ekleme ve silme gibi işlemler genellikle daha verimli olabilir.
2. **Düzensiz Diziler (Unsorted Arrays):**
   * Elemanları belirli bir düzene göre sıralanmamış dizilerdir.
   * Elemanlar herhangi bir sırada olabilir.
   * Bu tür dizilerde işlemler genellikle daha az verimli olabilir çünkü belirli bir düzen olmadığı için arama, ekleme ve silme işlemleri daha fazla zaman alabilir.

Dolayısıyla, diziler hem düzenli hem de düzensiz olarak tanımlanabilirler. Bu, dizi elemanlarının sıralı olup olmamasına bağlıdır.



ChatGPT

Tek yönlü bağlı listede sondan eleman silmenin maliyeti 𝑂(1)*O*(1) değildir. Bu ifade yanlıştır (false).

**Açıklama:** Tek yönlü bağlı listede bir elemanı sondan silmek için, sondaki düğümün öncesindeki düğümün (yani sondan bir önceki düğümün) bulunması gereklidir. Bu, genellikle listenin başından başlayarak düğümleri tek tek dolaşıp sondan bir önceki düğüme ulaşmayı gerektirir. Listenin her bir düğümünü kontrol ederek sondan bir önceki düğüme ulaşmak 𝑂(𝑛)*O*(*n*) zaman alır, çünkü n elemanı içeren bir listede n-1 düğümü kontrol etmeniz gerekebilir.

Bu nedenle, tek yönlü bağlı bir listede sondan eleman silmenin zaman karmaşıklığı 𝑂(1)*O*(1) değil, 𝑂(𝑛)*O*(*n*) olur.

Aşağıdakilerden hangisi Bağlı Listelerin avantajlarından biri değildir? Liste başına ekleme işlevi sabit zamanda gerçekleştirilir. Liste elemanları üzerinde dolaşmanın maliyeti O(n)'dir. Liste başından silme işlevi sabit zamanda gerçekleştirilir. Dinamik bellek yönetimi yapılmasına olanak verir.

ChatGPT

Bağlı listelerin avantajlarından biri olmayan ifade şudur:

**Liste elemanları üzerinde dolaşmanın maliyeti O(n)'dir.**

Bu ifade, bağlı listelerin bir avantajı değil, bir sınırlamasıdır. Bağlı listelerin avantajları genellikle şunlardır:

* **Liste başına ekleme işlevi sabit zamanda gerçekleştirilir (O(1)):** Bağlı listelerde, bir elemanı listenin başına eklemek sabit zamanda yapılabilir.
* **Liste başından silme işlevi sabit zamanda gerçekleştirilir (O(1)):** Listenin başındaki bir elemanı silmek de sabit zamanda yapılabilir.
* **Dinamik bellek yönetimi yapılmasına olanak verir:** Bağlı listeler, bellek yönetimi açısından esneklik sağlar ve gerektiğinde belleği dinamik olarak ayırıp serbest bırakabilir.

Ancak, liste elemanları üzerinde dolaşmanın maliyeti 𝑂(𝑛)*O*(*n*) olması, bağlı listelerin bir avantajı değil, dezavantajıdır. Bağlı listelerde elemanlar üzerinde dolaşmak için listenin başından sonuna kadar sırasıyla ilerlemek gerekir, bu da zaman karmaşıklığı açısından 𝑂(𝑛)*O*(*n*) maliyetlidir.

Diziler sabit boyutlu olarak tanımlanır. false true

ChatGPT

"Diziler sabit boyutlu olarak tanımlanır." ifadesi **doğrudur** (true).

**Açıklama:** Diziler, genellikle tanımlandıkları anda belirlenen ve daha sonra değiştirilemeyen sabit bir boyuta sahiptir. Bu, dizilerin bellek alanı yönetiminde belirli bir alanın önceden ayrılması anlamına gelir. Sabit boyutlu dizilerde (statik diziler), bir kez oluşturulduktan sonra dizinin boyutu değiştirilmez.

Örneğin, C ve Java gibi birçok programlama dilinde bir dizi tanımlandığında, bu dizinin boyutu sabit olur:

c

Copy code

int myArray[10]; // Boyutu 10 olan bir dizi

Bu dizinin boyutu 10 olarak tanımlanmıştır ve daha sonra bu boyut değiştirilemez.

Dinamik diziler veya dinamik olarak tahsis edilen bellek yapıları farklıdır ve bunların boyutu programın çalışması sırasında değiştirilebilir. Ancak, genel olarak ve temel tanımlamaya göre, diziler sabit boyutlu olarak tanımlanır.

Dizi boyutunun dinamik olarak yönetilmesi isteniyor ise ArrayList ya da Generic List kullanılabilir. true false

ChatGPT

"Dizi boyutunun dinamik olarak yönetilmesi isteniyor ise ArrayList ya da Generic List kullanılabilir." ifadesi **doğrudur** (true).

**Açıklama:** Birçok modern programlama dilinde, standart diziler sabit boyutludur ve tanımlandıktan sonra boyutları değiştirilemez. Ancak, dinamik olarak büyüyebilen ve küçülebilen veri yapıları vardır. Örneğin:

* **Java'da ArrayList:** Java'da ArrayList, dinamik olarak boyutunu ayarlayabilen bir dizidir. Eleman ekledikçe veya çıkardıkça, ArrayList otomatik olarak yeniden boyutlandırılır.

java

Copy code

ArrayList<Integer> list = new ArrayList<>(); list.add(1); // Listeye eleman ekler list.remove(0); // Belirli bir indeksten eleman siler

* **C#'ta List<T>:** C# dilinde Generic List (List<T>), benzer şekilde dinamik boyutlandırma sağlar.

csharp

Copy code

List<int> list = new List<int>(); list.Add(1); // Listeye eleman ekler list.RemoveAt(0); // Belirli bir indeksten eleman siler

Bu veri yapıları, altında yatan diziyi gerektiğinde yeniden boyutlandırarak çalışır, böylece kullanıcıların boyutla ilgili endişelenmesine gerek kalmaz. Bu nedenle, dinamik boyut yönetimi gereken durumlarda ArrayList (Java) veya Generic List (C#) gibi yapıların kullanılması doğrudur.

Tek yönlü bağlı listede liste başından eleman silmenin maliyeti doğrusaldır. false true

ChatGPT

"Tek yönlü bağlı listede liste başından eleman silmenin maliyeti doğrusaldır." ifadesi **yanlıştır** (false).

**Açıklama:** Tek yönlü bağlı listede liste başından eleman silmenin maliyeti sabit zamandır (O(1)). Liste başındaki elemanı silmek için yalnızca baş işaretçisini bir sonraki düğüme ayarlamak yeterlidir. Bu işlem, liste ne kadar büyük olursa olsun sabit zamanda gerçekleşir. Dolayısıyla, maliyeti doğrusal (O(n)) değil, sabit (O(1)) zaman karmaşıklığına sahiptir.

markdown

Copy code

Örneğin: Liste: 1 -> 2 -> 3 -> 4 Baş işaretçisi (head) 1 düğümünü gösteriyor. Silme işlemi: head = head.next; Yeni Liste: 2 -> 3 -> 4 Baş işaretçisi artık 2 düğümünü gösteriyor. Bu işlem sabit zamanda gerçekleşir.

Tek yönlü bağlı listede arama yapmanın maliyeti O(1)'dir. false true

ChatGPT

"Tek yönlü bağlı listede arama yapmanın maliyeti O(1)'dir." ifadesi **yanlıştır** (false).

**Açıklama:** Tek yönlü bağlı listede bir elemanı aramak için genellikle listenin başından başlayarak her bir düğümü sırasıyla kontrol etmek gerekir. Bu işlem, aranan elemanın konumuna bağlı olarak en kötü durumda listenin sonuna kadar devam edebilir. Dolayısıyla, arama işleminin zaman karmaşıklığı O(n) olur, burada n listenin eleman sayısını temsil eder.

markdown

Copy code

Örneğin: Liste: 1 -> 2 -> 3 -> 4 Aranan eleman: 4 Arama işlemi: 1. İlk düğüm kontrol edilir, 1 değil. 2. İkinci düğüm kontrol edilir, 2 değil. 3. Üçüncü düğüm kontrol edilir, 3 değil. 4. Dördüncü düğüm kontrol edilir, 4. Bulundu! Bu işlem en kötü durumda O(n) zaman alır.

Bu nedenle, tek yönlü bağlı listede arama yapmanın maliyeti O(1) değil, O(n) zaman karmaşıklığına sahiptir.

Tek yönlü bağlı listede arama yapmanın maliyeti O(n)'dir. false true

ChatGPT

"Tek yönlü bağlı listede arama yapmanın maliyeti O(n)'dir." ifadesi **doğrudur** (true).

**Açıklama:** Tek yönlü bağlı listede bir elemanı aramak için genellikle listenin başından başlayarak her bir düğümü sırasıyla kontrol etmek gerekir. Bu işlem, aranan elemanın konumuna bağlı olarak en kötü durumda listenin sonuna kadar devam edebilir. Dolayısıyla, arama işleminin zaman karmaşıklığı O(n) olur, burada n listenin eleman sayısını temsil eder.

markdown

Copy code

Örneğin: Liste: 1 -> 2 -> 3 -> 4 Aranan eleman: 4 Arama işlemi: 1. İlk düğüm kontrol edilir, 1 değil. 2. İkinci düğüm kontrol edilir, 2 değil. 3. Üçüncü düğüm kontrol edilir, 3 değil. 4. Dördüncü düğüm kontrol edilir, 4. Bulundu! Bu işlem en kötü durumda O(n) zaman alır.

Bu nedenle, tek yönlü bağlı listede arama yapmanın maliyeti O(n) zaman karmaşıklığına sahiptir.

Top of Form