GİRESUN ÜNİVERSİTESİ MÜHENDİSLİK FAKÜLTESİ
BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ
VERİ YAPILARI DERSİ VİZE SINAVI

Adı – Soyadı – Numarası:

Soru 1: Bir dizinin elemanları bellekte ardışık olarak depolanır. Diziye rastgele bir konuma eleman eklemek veya çıkarmak neden zordur? Bu işlemler neden verimsizdir?

Diziler, bellekte ardışık (contiguous) olarak depolanan veri yapılarıdır. Bu nedenle dizilere rastgele bir konuma eleman eklemek veya çıkarmak belirli zorluklar ve verimsizlikler içerir.

Bellek Yapısı ve Erişim: Dizilerde, elemanlar bellekte birbiri ardına sıralanır. Yani, dizinin her elemanının bellekteki yeri sabittir ve belirli bir indeks ile doğrudan erişilebilir. Bu sabit yapının avantajı, hızlı erişim sağlamaktır. Ancak, bu durum aynı zamanda dizilere eleman eklemek veya çıkarmak için verimsiz bir süreç oluşturur.

Diziye Eleman Ekleme (Araya Ekleme): Eğer dizinin herhangi bir yerine eleman eklemek isterseniz, o noktadaki ve sonrasındaki elemanları birer birer kaydırmanız gerekir. Örneğin, dizinin ortasında bir elemana yeni bir eleman eklemek için şu işlemler yapılmalıdır: Öncelikle, eklemeyi düşündüğünüz yerden sonrasındaki tüm elemanlar bir konum sağa kaydırılır. Yeni eleman, boş olan konuma eklenir. Bu işlemin zaman karmaşıklığı O(n)'dir, çünkü ekleme işlemi dizinin yarısına veya tamamına kadar olan elemanları kaydırmak zorunda kalır (en kötü durumda).

Diziye Eleman Ekleme (Sonuna Ekleme): Eğer diziye sonuna ekleme yapıyorsanız ve dizinin kapasitesi önceden belirlenmişse, bir "kapasite aşımı" durumu oluşursa dizinin tüm elemanları yeni bir diziye taşınır ve ardından yeni eleman eklenir. Bu durumda da zaman karmaşıklığı O(n)'e çıkar, çünkü eski dizinin tüm elemanlarının kopyalanması gerekecektir.

Dizi Elemanını Çıkarmak (Aradan Çıkarma): Bir elemanı diziden çıkarmak için, çıkarılacak elemanın yerine gelen tüm elemanların sola kaydırılması gerekir. Yani, çıkarılan elemandan sonraki tüm elemanlar birer birer sola kaydırılır. Bu işlem yine O(n) zaman karmaşıklığına sahiptir, çünkü çıkarmadan sonra kalan tüm elemanların yer değiştirilmesi gerekir.

Dizi Elemanını Çıkarmak (Son Elemanı Çıkarmak): Eğer son eleman çıkarılıyorsa, bu işlem O(1) zaman karmaşıklığına sahiptir, çünkü dizinin sonundaki eleman kaldırılır ve herhangi bir elemanı kaydırmak gerekmez. Ancak, ortada bir eleman çıkarılması gerektiğinde, kalan elemanların kaydırılması zorunludur.

Dizilere rastgele erişim ve araya ekleme işlemleri O(n) zaman alır, bu da dizinin boyutunun büyüklüğüne bağlı olarak çok verimsiz hale gelir. Özellikle büyük dizilerde bu tür işlemler çok pahalı hale gelir.

Eğer dizi kapasitesine ulaştığında büyütülmesi gerekiyorsa, yeni bir dizi oluşturulup eski dizinin tüm elemanları bu yeni diziye kopyalanmalıdır. Bu işlemi her kapasite aşımında yapmak gerektiğinde önemli bir bellek ve işlemci maliyeti oluşur.

Soru 2: Bağlı liste üzerinde bir düğüm (node) silme işlemini açıklayınız. Silme işlemi sırasında neden bir önceki düğüm bilgisine ihtiyaç duyulur?

Bağlı listelerde her düğüm, veriyi ve bir sonraki düğümün adresini tutar. Düğüm silme işlemi sırasında, bu bağlantıların düzgün bir şekilde güncellenmesi gerekir.

Silinecek Düğümün Belirlenmesi: İlk olarak, silmek istediğiniz düğümün yeri belirlenir. Bu, genellikle listeyi gezerek ve uygun düğümü bulmaya çalışarak yapılır. Düğüm bulunduğunda, silme işlemi başlar.

Bağlantıların Güncellenmesi: Bir önceki düğümün bağlantısını güncelleme: Bağlı listelerde, her düğüm bir sonraki düğümün adresini tutar. Eğer bir düğüm silinecekse, silinecek düğümden önceki düğümün next işaretçisi, silinen düğümün bir sonraki düğümüne işaret etmelidir.

Başarılar.. Dr. Öğr. Üyesi Sercan KÜLCÜ 6.11.2024

Örneğin, A -> B -> C -> D şeklinde bir bağlı liste varsa ve B düğümünü silmek istiyorsak, A düğümünün next işaretçisi C düğümünü işaret etmelidir. Yani, A.next = C yapılır.

Baş Düğüm Silme (Head node): Eğer silmek istenen düğüm baş düğümse (örneğin, A), listeyi gezmek gerekmez, çünkü baş düğüm doğrudan head işaretçisi tarafından tutulur. Baş düğüm silindiğinde, baş işaretçisi yeni başa, yani listenin ikinci elemanına (varsa) işaret edecek şekilde güncellenir.

Son Düğüm Silme: Eğer son düğüm siliniyorsa, önceki düğümün next işaretçisi null'a işaret etmelidir, çünkü son düğümün ardından başka bir düğüm yoktur.

Soru 3: Dairesel (circular) kuyruğun klasik kuyruktan farkı nedir? Dairesel kuyruğun avantajını bir örnek ile açıklayınız.

Klasik Kuyruk (Linear Queue):

Klasik kuyruklar, bir ilk giren, ilk çıkar (FIFO) veri yapısıdır. Elemanlar kuyruğa eklenirken (enqueue), kuyruğun sonuna eklenir ve elemanlar kuyruğun başından çıkarılır (dequeue).

Ancak klasik kuyrukta, belirli bir kapasite (önceden tanımlı dizi boyutu) vardır ve bu kapasite dolduğunda yeni eleman eklenemez.

Bellek israfı: Kuyruğun başından elemanlar çıkarıldığında, bu elemanların yerleri boşalmış olur ve kuyruğun sonunda yer açıldığı halde, baş kısmındaki boş alan kullanılamaz. Bu durumda bellek israfı meydana gelir. Yani, baştaki boş alan yeniden kullanılmaz.

Dairesel Kuyruk (Circular Queue):

Dairesel kuyruklar, tıpkı klasik kuyruklar gibi FIFO prensibiyle çalışır, ancak önemli farkı, kuyruk başı ve kuyruk sonu arasındaki bağlantının dairesel şekilde düzenlenmesidir.

Dairesel kuyruğun başı ve sonu birbirine bağlanır. Yani, kuyruğun sonuna gelindiğinde, kuyruk bir döngüye girer ve baştan devam eder. Bu sayede kuyruktaki boş alanlar daha verimli kullanılabilir.

Bellek verimliliği: Kuyruğun sonu, başa bağlandığı için, baştaki boş alanlar yeniden kullanılabilir. Bu, kuyruk kapasitesinin etkin kullanımını sağlar.

Soru 4: Aşağıda verilen sınıf tanımında, dizi yapısı kullanılarak Yığıt (Stack) veri yapısı geliştirmek için gerekli cikar (pop) metodunu yazınız.

```
public class YigitDiziGosterimi {

	private int kapasite;
	private int tepe;
	private int[] dizi;

	public int cikar() {
		if (tepe == -1) {
			System.out.println("Yığın boş. Çıkarma yapılamaz.");
			return -1; // Boş yığın için bir belirleyici değer döndürün.
		} else {
			int cikarilanDeger = dizi[tepe--];
			System.out.println(cikarilanDeger + " yığından çıkarılmıştır.");
			return cikarilanDeger;
		}
	}
}
```

Soru 5: Aşağıda verilen sınıf tanımında, bağlı liste (linkedlist) yapısı kullanılarak Kuyruk (Queue) veri yapısı geliştirmek için gerekli ekle (enqueue) metodunu yazınız.

```
public class KuyrukBagliListeGosterimi<E> {

	private TekYonluDugum<E> bas;
	private TekYonluDugum<E> son;
	private int boyut;

	public void ekle(E eleman) {
		// Yeni bir TekYonluDugum öğesi oluştur
		TekYonluDugum<E> yeniDugum = new TekYonluDugum<>(eleman);

		// Eğer liste boşsa (baş değişkeni null ise), yeni düğümü başa ekle
		if (boyut == 0) {
			bas = yeniDugum;
			son = yeniDugum;
		} else {
			// Eğer liste boş değilse, son düğümün sonraki referansını yeni düğüme
bağla ve son düğümü güncelle
			son.sonraki = yeniDugum;
			son = yeniDugum;
		}
		// Liste boyutunu bir artır
		boyut++;
	}
}
```