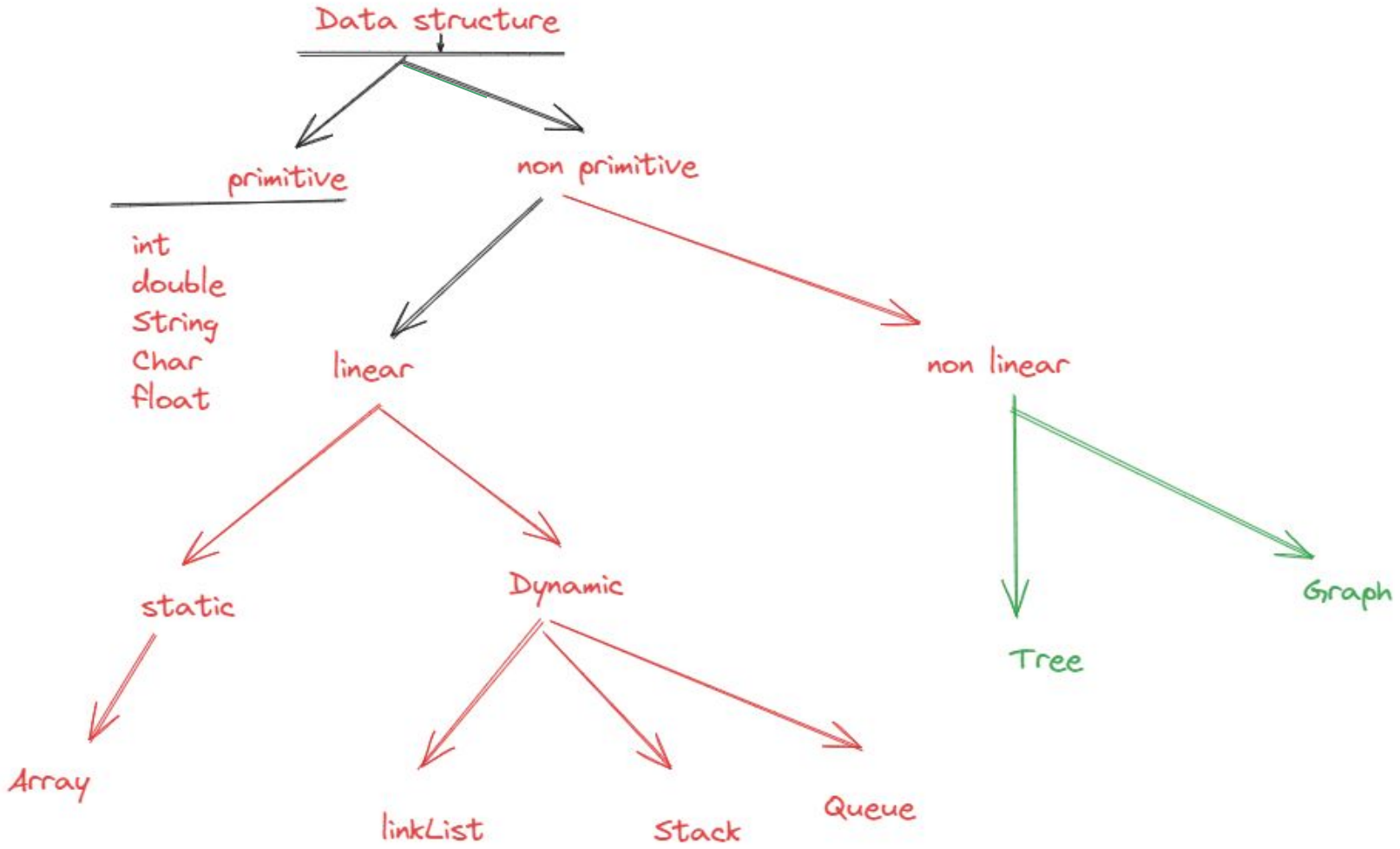
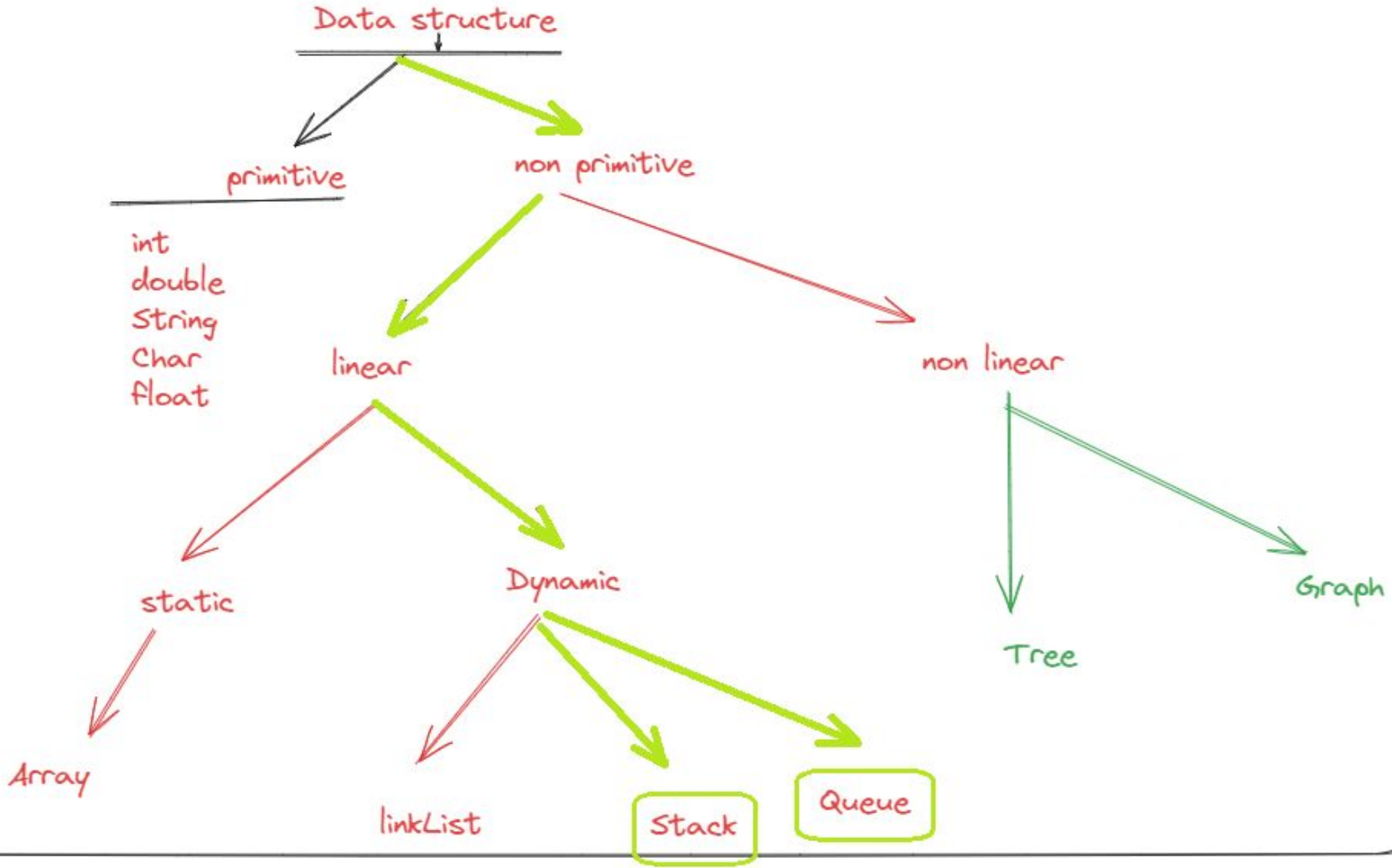
Kuyruk Ve Yığıt Yapılarının Kullanımı

Melike KUTLU 2211502003
Osman ARI 2211502018
Rabia ÖZDEMİR 2211502021
Şerife ENGİNAR 2211502047

Sunumun Amacı

Bu sunumda kuyruk(queue) ve yığıt(stack) yapılarının işleyişi, kullanım alanları, java kodu ve avantajları incelenecektir.


Data structure
primitive
non primitive
int
double
String
Char
float
linear
non linear
static
Dynamic
Graph
Tree
Array
linkList
Stack
Queue

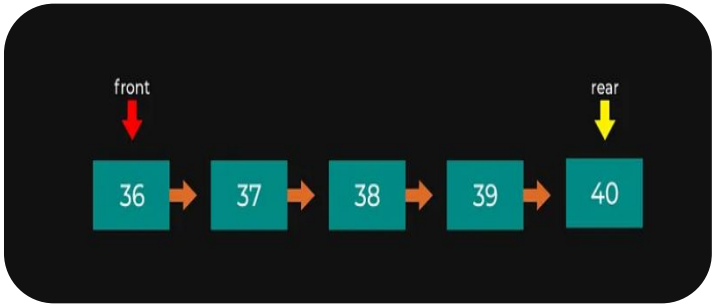

Data structure
primitive
non primitive
int
double
String
Char
float
linear
non linear
static
Dynamic
Graph
Tree
Array
linkList
Stack
Queue

Konu Başlıkları

- Kuyruk(Queue) Yapısı
- Yığıt(Stack) Yapısı
- Kullanım Alanları
- Kod örnekleri-Algoritma Analizleri
- Yığıt ve Kuyruk Yapılarının Avantaj ve Dezavantajları
- Sonuç
- Soru-Cevap

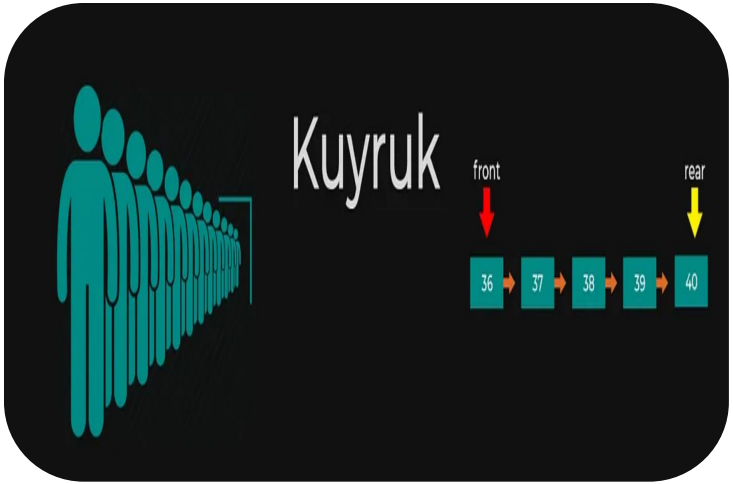
KUYRUK (QUEUE)

- Kuyruklar, eleman eklemelerinin sondan (rear) ve eleman çıkarmalarının baştan (front) yapıldığı doğrusal veri yapılarıdır.



- Bir eleman ekleneceği zaman kuyruğun sonuna eklenir.
- Bir eleman çıkarılacağı zaman kuyrukta bulunan ilk eleman çıkarılır.

- Bu nedenle kuyruklara FIFO (First In First Out-İlk giren ilk çıkar) veya LILO (Last-in-Last-out-Son giren son çıkar) listeleri de denilmektedir.



Ana Kuyruk İşlemleri:

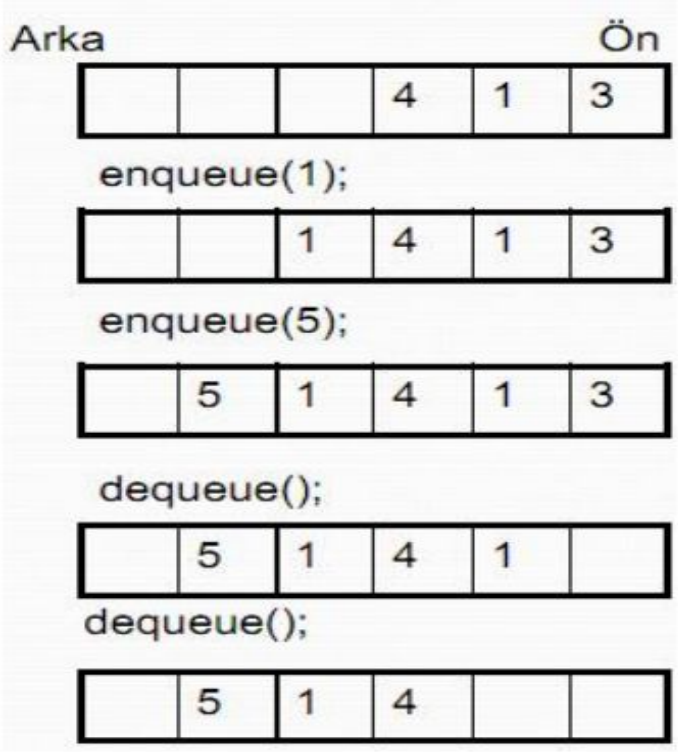
İki tane temel işlem yapılabilir ;

- **enqueue (object)**, bir nesneyi kuyruğun en sonuna ekler (insert).
- **dequeue (object)**, Kuyruk başındaki nesneyi getirir ve kuyruktan çıkarır (remove veya delete).

Yardımcı kuyruk işlemleri:

- **object front()** (getHead/getFront): kuyruk başındaki nesneyi kuyruktan çıkarmadan geri döndürür.
- **integer size()**: kuyrukta saklanan nesne sayısını geri döner
- **boolean isEmpty()**: Kuyrukta nesne olup olmadığını kontrol eder.

KUYRUK EKLEME / ÇIKARMA



KUYRUK (QUEUE) KOD YAPISI

KUYRUK METOTLARI

Add(): Kuyruğa yeni bir eleman ekler.

Peek(): Kuyruğun en önündeki elemanı silmeden döndürür.

Element(): Peek() ile aynı işlevi gerçekleştirir.

Remove(): Kuyruğun başındaki elemanı siler ve bu değeri döndürür.

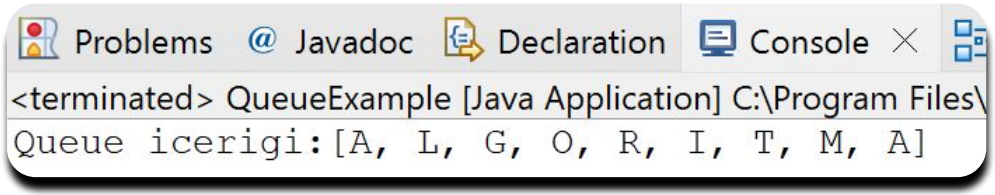
Poll(): Kuyruğun başındaki elemanı siler ve kalan ilk elemanı döndürür.

Size(): Kuyruğun eleman sayısını döndürür.

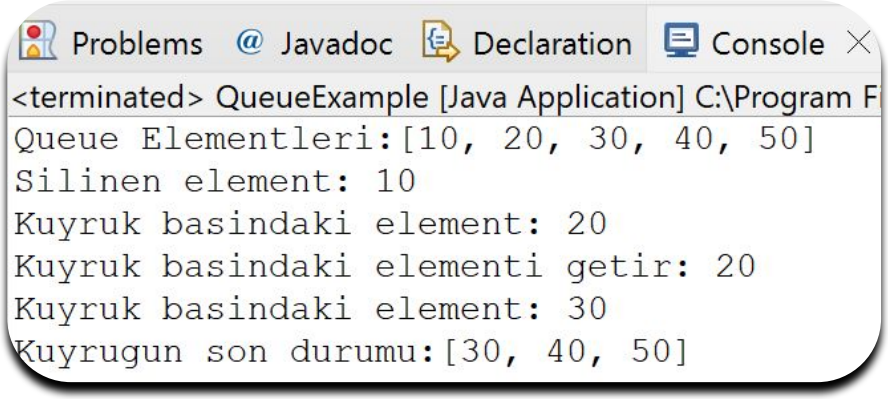
```
Queue oluştur:
    yeni bir boş dizi oluştur
    kuyruğun başını ve sonunu temsil eden indeksi belirle(baş = 0, son = -1)
    elemanSayısı = 0
Enqueue(eleman):
    son = son + 1
    kuyruk[son] = eleman
    elemanSayısı = elemanSayısı + 1
Dequeue():
    eğer elemanSayısı = 0 ise:
        "Kuyruk boş, eleman çıkartılamaz" mesajını göster
        çık
    eleman = kuyruk[baş]
    baş = baş + 1
    elemanSayısı = elemanSayısı - 1
    elemanı döndür
Front():
    eğer elemanSayısı = 0 ise:
        "Kuyruk boş, başa bakılamaz" mesajını göster
        çık
    eleman = kuyruk[baş]
    elemanı döndür
KuyrukBoşMu():
    eğer elemanSayısı = 0 ise:
        doğru değeri döndür
    değilse:
        yanlış değeri döndür
```

PSEUDO CODE

```java
import java.util.LinkedList;
import java.util.*;
public class QueueExample {

    public static void main(String[] args) {

        Queue<String> kuyruk = new LinkedList<String> ();

        kuyruk.add("A");
        kuyruk.add("L");
        kuyruk.add("G");
        kuyruk.add("O");
        kuyruk.add("R");
        kuyruk.add("I");
        kuyruk.add("T");
        kuyruk.add("M");
        kuyruk.add("A");


        System.out.println("Queue icerigi:" + kuyruk);
```



```java
import java.util.LinkedList;
import java.util.*;
public class QueueExample {

    public static void main(String[] args) {

        Queue<Integer> q1 = new LinkedList<Integer>();
        //add() method
        q1.add(10);
        q1.add(20);
        q1.add(30);
        q1.add(40);
        q1.add(50);

        System.out.println("Queue Elementleri:"+q1);

        //remove () method
        System.out.println("Silinen element: "+q1.remove());

        //element() queue başındaki elementi getir
        System.out.println("Kuyruk basindaki element: "+q1.element());

        //poll () elementi sil ve getir
        System.out.println("Kuyruk basindaki elementi getir: "+q1.poll());

        //queue başındaki elementi getir
        System.out.println("Kuyruk basindaki element: "+q1.peek());

        //print Queue elementlerini yazdır
        System.out.println("Kuyrugun son durumu:"+q1);
```



Kuyruk oluşturma ve kuyruğa eleman ekleme:

```
public class Queue {
        private static int front, rear, capacity;
        private static int queue[];

        Queue(int size) {
            front = rear = 0;
            capacity = size;
            queue = new int[capacity];
        }
        // Kuyruğa bir element ekler
        static void queueEnqueue(int item)  {
          // Kuyruk dolulugunu kontrol eder
          if (capacity == rear) {
              System.out.printf("\nQueue Dolu\n");
              return;
          }
          // Kuyruğun sonuna eleman ekler
          else {
              queue[rear] = item;
              rear++;
          }
          return;
        }
```

Kuyruktan bir eleman silme:

```
//Kuyruktan eleman siler
static void queueDequeue()  {
  // check if queue is empty
  if (front == rear) {
      System.out.printf("\nQueue boş\n");
      return;
  }
  // elemanları arkaya kaydırır
  else {
      for (int i = 0; i < rear - 1; i++) {
          queue[i] = queue[i + 1];
      }
//queue[rear]  0 yap
      if (rear < capacity)
          queue[rear] = 0;

      // rear değerini bir azaltır
      rear--;
  }
  return;
```

Kuyruğu ekrana yazdırmak ve ilk elemanı döndürme:

```
// kuyruk elementlerini gösterir
static void queueDisplay()
{
  int i;
  if (front == rear) {
      System.out.printf("Queue Boş\n");
      return;
  }
  // önden arkaya elementleri yazdırır
  for (i = front; i < rear; i++) {
      System.out.printf(" %d = ", queue[i]);
  }
  return;
}
// ilk elemanı yazdırır
static void queueFront()
{
    if (front == rear) {
        System.out.printf("Queue boş\n");
        return;
    }
    System.out.printf("\n Queue başındaki elementi yazdır: %d", queue[front]);
    return;
}
```

```java
public static void main(String[] args) {
  // kapasitesi 4 olan kuyruk oluşturduk
  Queue q = new Queue(4);

 // Queue elementlerini yazdır
  q.queueDisplay();
  // sıraya eleman ekleme
  q.queueEnqueue(10);
  q.queueEnqueue(30);
  q.queueEnqueue(50);
  q.queueEnqueue(70);
  // Elementleri yazdır
  System.out.println(" ekleme yapildiktan sonra:");
  q.queueDisplay();
  // queue başındaki elementi yazdır
  q.queueFront();
  // queue element ekle
  q.queueEnqueue(90);
  // queue elementlerini yazdır
  q.queueDisplay();
  q.queueDequeue();
  q.queueDequeue();
  System.out.printf("\nkuyruktan iki eleman cikinca:");

  // Queue elementlerini yazdır
  q.queueDisplay();
  // queue başındaki elementi yazdır
  q.queueFront();
```

Console

```
<terminated> Queue [Java Application] C:\Program Files\Java\jd
Queue Bos
ekleme yapildiktan sonra:
10 - 30 - 50 - 70 -
Queue basindaki elementi yazdir: 10
Queue Dolu
10 - 30 - 50 - 70 -
kuyruktan iki eleman cikinca:50 - 70 -
Queue basindaki elementi yazdir: 50
```

BİG O(1) =SABİT ZAMAN KARMAŞIKLIĞI

(GENELLİKLE)

KUYRUK ÖRNEKLERİ

Ağ yönetimi: Bir ağdaki veri paketlerinin iletimi için kullanılır. Paketler, kuyruğa eklenir ve sırayla işlenir.

İşletim sistemi işlem yönetimi: İşletim sistemleri, CPU'ya gelen işlemleri kuyruğa alıp sırayla işleyerek işlem yönetimini sağlar.

Bellek yönetimi: Bir işletim sistemi, bellek bloklarının tahsis edilmesi ve serbest bırakılması için kullanılabilir.

Kaynak Tahsisi: Bir dosya sunucusu, birden çok kullanıcının dosya isteklerini işlemek zorunda olduğunda, dosya istekleri kuyruğa eklenir ve sırayla işlenir.

KUYRUK ÖRNEKLERİ

Müzik ve video oynatıcılar: Müzik veya video oynatıcılar, çalma listelerini veya oynatma sırasını yönetmek için kullanılabilir.

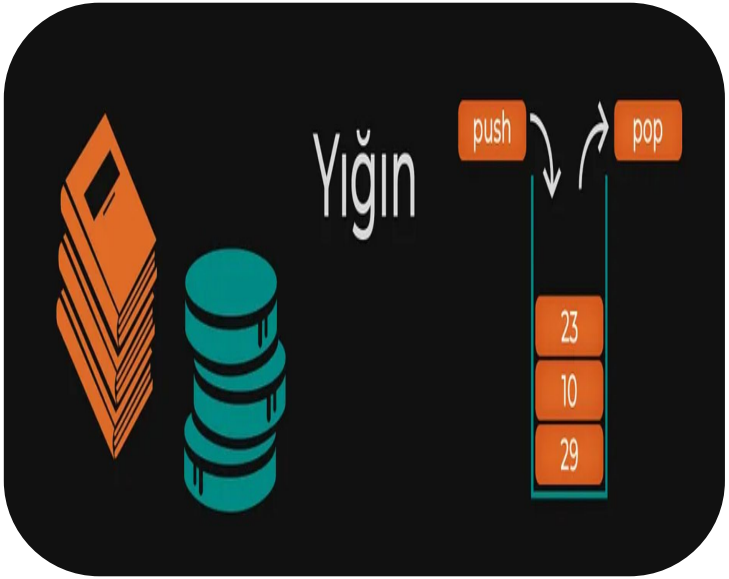
Çağrı merkezleri ve müşteri hizmetleri: Müşteri temsilcilerinin kuyruktaki çağrıları sırayla yanıtlamasını sağlar.

Baskı kuyrukları: Birden fazla kullanıcının aynı yazıcıya iş göndermesi durumunda kullanılabilir.

Breadth-First Search (BFS):Geniş öncelikli arama algoritmalarında grafikleri veya ağaçları seviye seviye geçmek için kullanılır.

YIĞIT (STACK)

- Son giren ilk çıkar (Last In First Out-LIFO) veya İlk giren son çıkar (First-in-Last-out FILO) mantığıyla çalışır.
- Eleman ekleme çıkarmalarının en üstten (top) yapıldığı veri yapısına yığıt (stack) adı verilir.
- Bir eleman ekleneceği zaman, yığıtın en üstüne konulur.
- Bir eleman çıkarılacağı zaman, yığıtın en üstündeki eleman çıkarılır.
- Bu eleman da yığıttaki elemanlar içindeki en son eklenen elemandır. Bu nedenle yığıtlara LIFO (Last In First Out Son giren ilk çıkar) listesi de denilir.



Yığın yapısını gerçekleştirmek için 2 yol vardır.

- Dizi kullanmak
- Bağlantılı liste kullanmak

- **empty stack:** Boş yığıt
- **push (koy):** Yığıta eleman ekleme
- **pop (al):** Yığıttan eleman çıkarma

YIĞIN İŞLEMLERİ

Ana yığın işlemleri:

push(nesne): yeni bir nesne ekler

Girdi: Nesne Çıktı: Yok

pop(): en son eklenen nesneyi çıkarıp geri döndürür.

Girdi: Yok Çıktı: Nesne

Yardımcı yığın işlemleri:

top(): en son eklenen nesneyi çıkarmadan geri döndürür.

Girdi: Yok Çıktı: Nesne

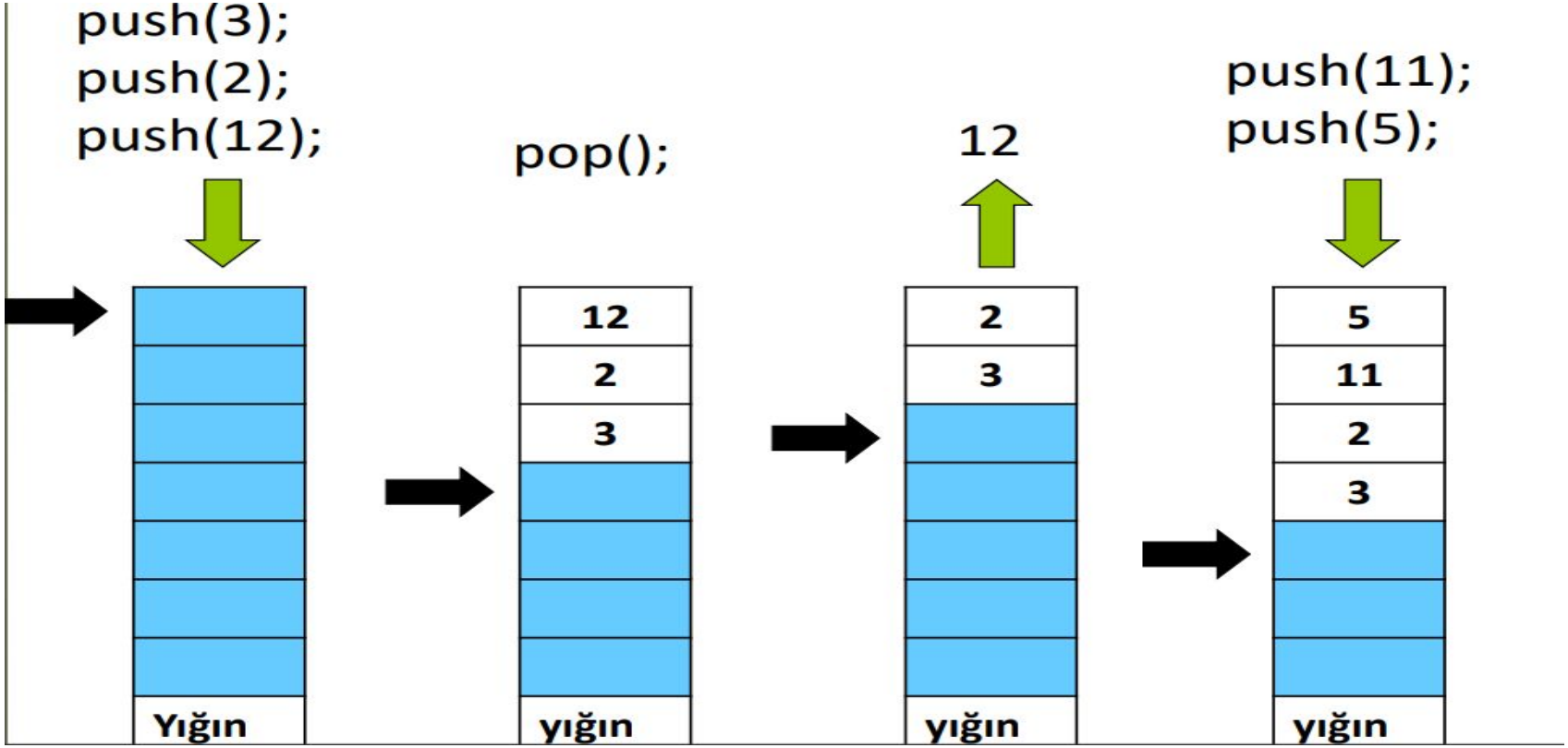
size(): Bu işlev, yığında depolanan öğe sayısını döndürür. Yani yığıtın boyutunu verir.

Girdi: Yok Çıktı: Tamsayı

isEmpty(): yığında nesne bulunup bulunmadığı bilgisi geri döner. Yani yığıtın boş olup olmadığını kontrol eder.

Girdi: Yok Çıktı: Boolean

YIĞIN (STACK) YAPISI



YIĞIT(STACK) KOD YAPISI

YIĞIT METOTLARI

Push: Yığına yeni bir eleman ekler

Pop: Yığıttan bir eleman siler.

Peek: Yığıtın en üstündeki elemanı döndürür ancak silmez.

IsEmpty: Yığıtın boş olup olmadığını kontrol eder.

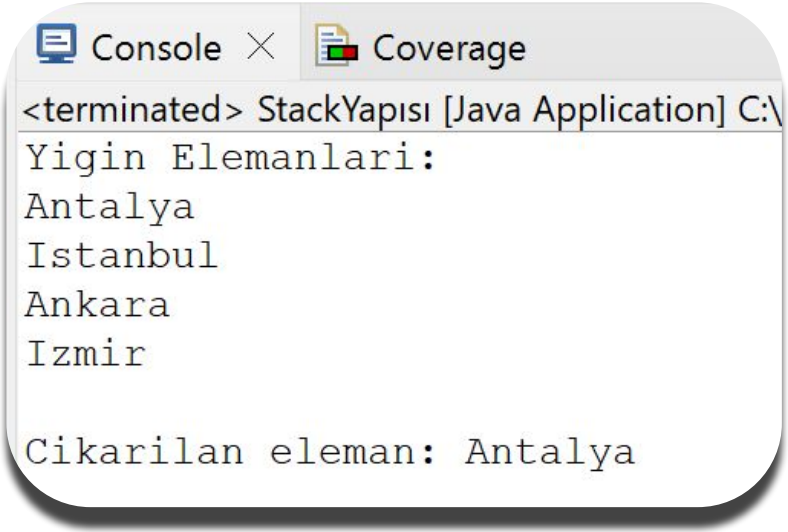
```
Stack oluştur:
    yeni bir boş dizi oluştur
    yığıtın üstünü temsil eden bir indeks belirle (örneğin, top = -1)

Push(eleman):
    top = top + 1
    stack[top] = eleman
Pop():
    eğer top < 0 ise:
        "Yığıt boş,eleman çıkartılamaz" mesajını göster
        çık
    eleman = stack[top]
    top = top - 1
    elemanı döndür
Peek():
    eğer top < 0 ise:
        "Yığıt boş,elemana bakılamaz" mesajını göster
        çık
    eleman = stack[top]
    elemanı döndür
YığıtBoşMu():
    eğer top < 0 ise:
        doğru değeri döndür
    değilse:
        yanlış değeri döndür
```

PSEUDO CODE

Yığına eleman ekleme,silme ve yığını ekrana yazma:

```
public class StackYapısı {
    //String tipinde bir dizi tanımlıyoruz.
    //top ise dizinin değişken uzunluklu olmasını sağlıyor.
    String dizi[];
    int top=0;

    public  StackYapısı(int k){
    dizi=new String[k];
    }

    //diziye bir eleman ekler
    public void push(String veri){
    dizi[top]=veri;
    top++;
    }
    //diziden bir eleman çıkarır
    public String pop(){
        if(top>0)
            return dizi[--top];
        else
            return "Yigin bos";
    }
    //Elemanları ekrana yazdırır
    public void display() {
        if (top == 0) {
            System.out.println("Yigi Bos");
            return;
        }
        System.out.println("Yigin Elemanlari:");
        for (int i = top - 1; i >= 0; i--) {
            System.out.println(dizi[i]);
        }
    }
```

```java
public static void main(String[] args) {

    StackYapısı stack =new StackYapısı(10);

    stack.push("Izmir");

    stack.push("Ankara");

    stack.push("Istanbul");

    stack.push("Antalya");
    stack.display();

    System.out.println("\nCikarilan eleman: "+stack.pop());
}
```



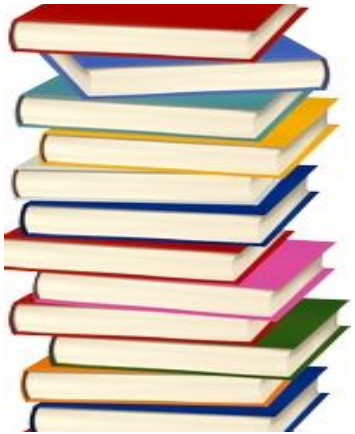
BİG O(1) =SABİT ZAMAN KARMAŞIKLIĞI

(GENELLİKLE)

Yığıt Kullanım Alanları

Günlük Hayattan Yığıt Örnekleri

- Üst üste konulan tabaklar,kitaplar,paralar...

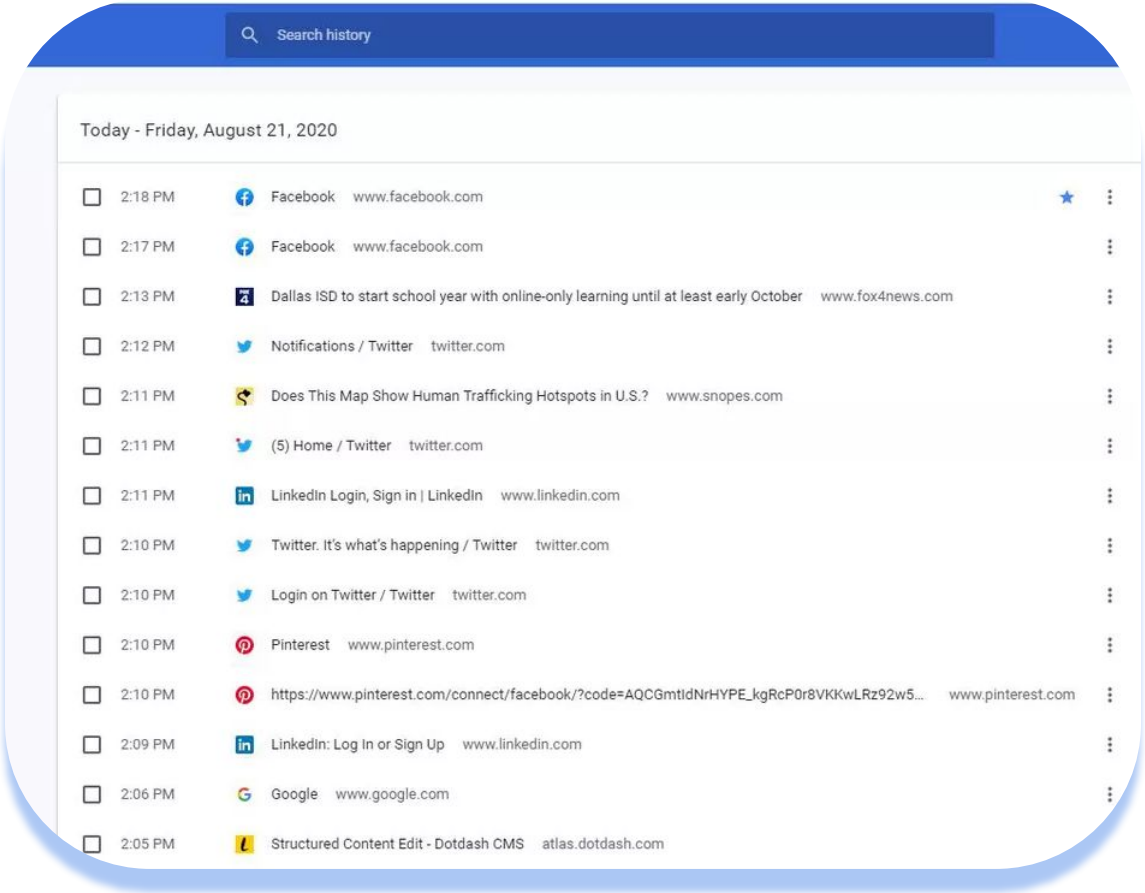


Bilgisayar Bilimlerinde Yığıt Örnekleri

- **Hafıza Yönetimi:**Programlar çalışırken geçici verilerin depolanması için kullanılır. Örneğin dinamik bellek tahsisi(malloc)için yığıt kullanılabilir.

- **Tarayıcı Geçmişi:**İnternet tarayıcıları,ziyaret edilen web sayfalarını yığıt yapısını kullanarak saklar(En son ziyaret edilen sayfa en üstte görünür).

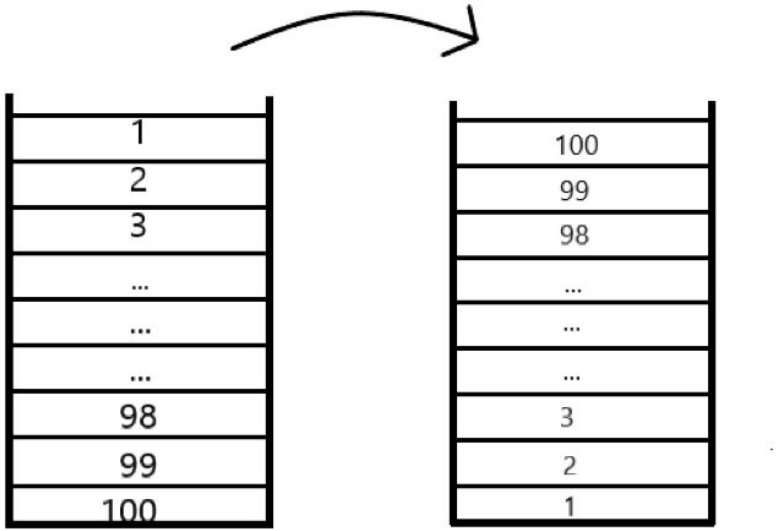
Tarayıcı Geçmişi Örneği



Yığıt Kullanım Örnekleri

- **Geri al/Yinele işlevleri**:Kullanıcının yaptığı işlemleri geri alma ve tekrar etmede kullanılır.
    - Art arda açılan web sayfalarında geri gelme işleminde yığıt yapısı kullanılır.
    - Ziyaret edilen sayfaların URL'leri bir yığıtta toplanır.Geri düğmesine basıldığında en sonda ziyaret edilen sayfalar görüntülenmek için yığıttaki URL'ler açılır.
- **Dönüştürme İşlemleri**:Bir veri yapısını tersine çevirme veya başka bir formata dönüştürmede kullanılabilir.

Dönüştürme İşlemi Örneği



Hanoi Kulelerinde Yığıt Kullanımı

- Her üç çubuk da birer yığıt olarak değerlendirilebilir.
- Amaç:Diskleri en sağdaki çubuğa taşımak
- İşlem sayısı:4 tane disk olduğu için 2^4=16. 16-1= 15 adımda gerçekleşir.

Şartlar:

- Aynı anda sadece 1 disk taşınabilir.
- Bir disk kendisinden daha küçük bir diskin üzerine taşınamaz.

Yığıt Kullanım Alanları

- **DFS(Depth First Search):**Derinlik öncelikli aramada bir düğümden başlanarak herhangi bir dal sonuna kadar takip edilir ve ardından geri dönülür.Aranan düğüm bulunana kadar arama işlemi aynı şekilde devam eder.Bu sırada gezilen düğümler yığıta eklenir.

- **Backtracking Algorithm:**Geri izleme algoritmasında işlem adımlarının takip edilmesinde yığıt yapısı kullanılabilir.Böylece yanlış bir adımda bir önceki adıma geri dönerek işlemlerin doğru şekilde devam ettirilmesi sağlanır.

Kuyruk Avantajları

- Büyük miktarda veri kolaylıkla ve verimli bir şekilde yönetilebilir.
- İlk giren ilk çıkar kuralına uyduğu için ekleme ve silme gibi işlemler kolaylıkla gerçekleştirebilir.
- Belirli bir hizmet birden fazla tüketici tarafından kullanıldığında kuyruklar kullanışlıdır.
- Kuyruklar, işlemler arası veri iletişimi için hızlıdır.
- Kuyruklar diğer veri yapılarının uygulanmasında kullanılabilir.

Kuyruk Dezavantajları

- Ortadan eleman ekleme ve çıkarma gibi işlemler zaman alır.
- Kuyruğa yeni bir öğe ancak mevcut öğeler kuyruktan silindiğinde eklenebilir.
- Bir elemanın aranması O(N) zaman alır.
- Bir dizi kullanarak uygulama yapıldığında kuyruğun maksimum boyutu önceden tanımlanmalıdır.

Yığıt Avantajları

- Basit ve anlaşılması kolay bir veri yapısıdır.
- Yığınlar üzerinde itme ve çekme işlemleri sabit zamanlı (O (1)) olarak gerçekleştirilebilir, bu da verilere verimli erişim sağlar.
- Fonksiyon çağrıları ve ifade değerlendirmesi gibi birçok durumda kullanılır.
- Geri alma/yenileme işlemleri

Yığıt Dezavantajları

- Yığındaki öğelere yalnızca üst kısımdan erişilebilir, bu da yığının ortasındaki öğelerin alınmasını veya değiştirilmesini zorlaştırır.
- Bir yığına tutabileceğinden daha fazla öğe itilirse, bir taşma hatası meydana gelir ve veri kaybına neden olur.
- Yığınlar öğelere rastgele erişime izin vermez, bu da onları öğelere belirli bir sırayla erişilmesi gereken uygulamalar için uygun hale getirmez.

Sonuç

- Kuyruk ve yığıtın günümüzdeki kullanım alanları:Veri yönetimi ve organizasyonu,işlem yönetimi,bellek yönetimi,paralel ve dağıtık hesaplama,algoritma tasarımı,ağ yönetimi,yapay zeka ve makine öğrenmesi,yorumlayıcılar,derleyiciler,fonksiyon çağrıları,tarayıcı geçmişi…
- Kuyruk ve yığıt yapıların kullanım alanı çok geniş olduğundan bu yapıları iyi bilmek birçok konunun anlaşılmasında kolaylık sağlayacaktır.
- Sonuç olarak kuyruk ve yığıt yapıları,bilgisayar bilimlerindeki temel yapı taşlarındandır.

Kaynakça

- https://github.com/themanoftalent
- https://openai.com/chatgpt
- https://www.studysmarter.co.uk/explanations/computer-science/data-structures/stack-in-data-structure/
- https://www.youtube.com/watch?v=vTI7CuYCtds
- https://medium.com/algorithms-data-structures/hanoi-kuleleri-1-a74bf4e7a48f
- https://www.lifewire.com/thmb/IKT5RT1pPh6vzuCyH8FN-b2CYx0=/1500x0/filters:no_upscale():max_bytes(150000):strip_icc():format(webp)/Capture-f464c0a6fdd24928ac5c0aea2ad777f9.JPG
- https://www.geeksforgeeks.org/
- https://medium.com/code-writers/backtracking-algorithms-using-stack-data-structure-d5de77c104c3#4db1

Dinlediğiniz İçin Teşekkürler