

CENG 201 Veri Yapıları 2: Yığıtlar ve Kuyruklar

Öğr.Gör. Şevket Umut ÇAKIR

Pamukkale Üniversitesi

Hafta 2

Anahat

① Yığıtlar(Stacks)

② Kuyruklar(Queues)

③ İşlem Notasyonları

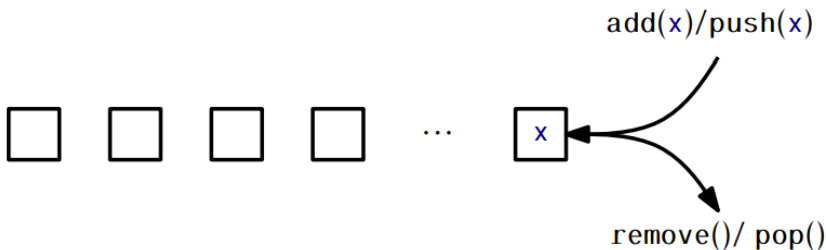
Soyut Veri Tipleri(SVT)

- Bir dizi işlem içeren nesnelerdir
- Tanımında işlemlerin **nasıl** yapıldığı belirtilmeyen matematiksel soyutlamalardır
- İşlemlerin gerçekleştirilmesinde(implementation) bir değişiklik olması durumunda son kullanıcı bundan etkilenmez
- Set(küme) SVT için *add*, *remove*, *size*, *contains*, *union* ve *find* işlemleri bulunur
- Listeler, kümeler, graflar SVT'lere örnek gösterilebilir

Yığıt Özellikleri

- Mutfak rafına üst üste dizilen tabaklar gibi düşünülebilir
- İçerisine farklı tiplerdeki verileri ekleyip çekebildiğimiz bir soyut veri yapısı
- Son giren ilk çıkar (Last In First Out, LIFO) mantığına göre çalışır
- Ekleme *push*, silme *pop* ve ilk elemanı görme *peek* işlevlerine sahiptir

Yığıt Görseli



Soru

- Aşağıdaki işlemler sonucunda yığıtın son hali ne olur?

```
push 5  
push 2  
push 7  
pop  
push 6  
push 9  
push 4  
pop  
pop  
push 12
```

Cevap



(a)
push
5
2
7

Cevap



(a)
push
5
2
7

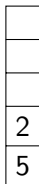


(b)
pop
→
7

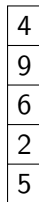
Cevap



(a)
push
5
2
7



(b)
pop
→
7



(c)
push
6
9
4

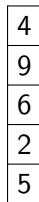
Cevap



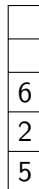
(a)
push
5
2
7



(b)
pop
→
7

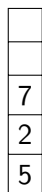


(c)
push
6
9
4

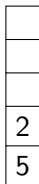


(d)
pop
→
4,
pop
→
9

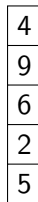
Cevap



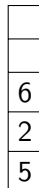
(a)
push
5
2
7



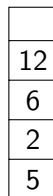
(b)
pop
→
7



(c)
push
6
9
4



(d)
pop
→
4,
pop
→
9



(e)
push
12

Figure: Cevap

Gerçekleştirme(Implementation)

- Yığın soyut bir veri tipidir
- Farklı veri yapıları kullanılarak gerçekleştirilebilir(Dizi, bağlı liste vb.)
- Ekleme *push*, silme *pop* ve ilk elemanı görme *peek* işlevlerini içermesi gerekir
- Hangi tip verileri saklayacak(integer, string, double)? Generic?
- Dizi kullanılarak nasıl gerçekleştirilebilir?

MyStack.java I

```
1 public class MyStack<T> {  
2     private T[] dizi;  
3     private int es;//eleman sayısı  
4  
5     public MyStack() {  
6         dizi=(T[]) new Object[10];  
7         es=0;  
8     }  
9  
10    public MyStack(int es) {  
11        dizi=(T[]) new Object[es];  
12    }  
13  
14    public void push(T eleman) {  
15        if(es==dizi.length)  
16            throw new RuntimeException("Stack overflow");  
17        dizi[es++]=eleman;  
18    }
```

MyStack.java II

```
20 public T pop() {
21     if(es==0)
22         throw new RuntimeException("Stack underflow");//unchecked exception
23     return dizi[--es];
24 }
25
26 public T peek() {
27     if(es==0)
28         throw new RuntimeException("Stack underflow");
29     return dizi[es-1];
30 }
31
32 public boolean isEmpty() {
33     return es==0;
34 }
35
36 public void print() {
37     for (int i = es-1; i >= 0 ; i--) {
38         System.out.println(dizi[i]);
39     }
40 }
41 }
```

StackMain.java Test Programı

```
1 public class StackMain {
2     public static void main(String[] args) {
3         MyStack<Integer> s=new MyStack<>();
4         s.push(5);
5         s.push(2);
6         s.push(7);
7         s.pop();
8         s.push(6);
9         s.push(9);
10        s.push(4);
11        s.pop();
12        s.pop();
13        s.push(12);
14        s.print();
15    }
16 }
```

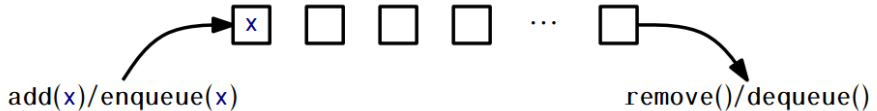
Kuyruk

- Gerçek hayattaki herhangi bir kuyruk/sıra gibidir
- İlk giren değer ilk çıkar(First In First Out, FIFO)
- Ekleme *enqueue* ve silme *dequeue* işlevlerini içermesi gerekir

Kuyruk Görseli

Interfaces

§1.2



Kuyruk örneği

- Aşağıdaki işlemler sonucunda yığının son hali ne olur?

```
enqueue 5  
enqueue 7  
enqueue 4  
dequeue  
dequeue  
dequeue  
enqueue 6
```

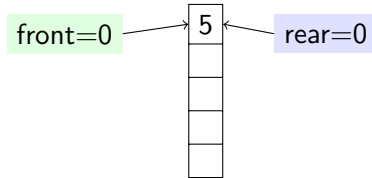


Figure: Enqueue 5

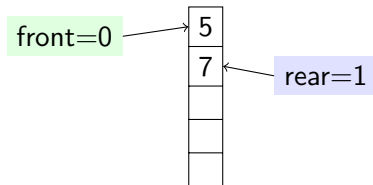


Figure: Enqueue 7

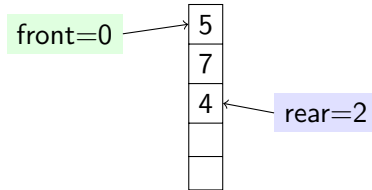


Figure: Enqueue 4

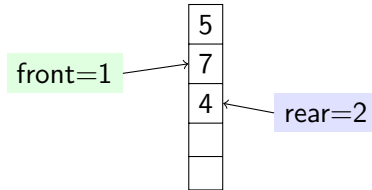


Figure: Dequeue → 5

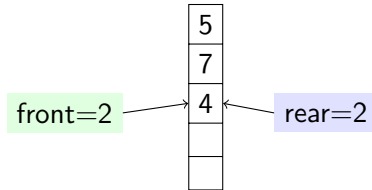


Figure: Dequeue → 7

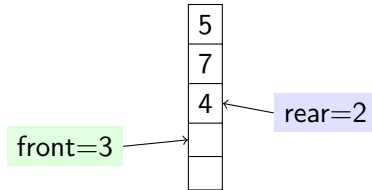


Figure: Dequeue \rightarrow 4, set `front=rear=-1`

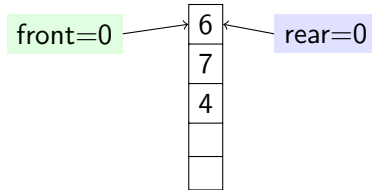


Figure: Enqueue 6

Kuyruk gerçekleştirilmesi

- Dizilerle gerçekleştirilecek
- Generic olacak

MyQueue.java I

```
1 public class MyQueue<T> {  
2     private int front=-1;  
3     private int rear=-1;  
4     T[] dizi;  
5  
6     public MyQueue() {  
7         dizi=(T[]) new Object[10];  
8     }  
9  
10    public MyQueue(int boyut) {  
11        dizi=(T[]) new Object[boyut];  
12    }  
13  
14    public boolean isEmpty() {  
15        return front==-1 && rear==-1;  
16    }  
17  
18    public boolean isFull() {  
19        return rear==dizi.length-1;  
20    }  
21 }
```

MyQueue.java II

```
22 public void enqueue(T eleman) {
23     if(isFull())
24         throw new RuntimeException("Queue is full");
25     if(isEmpty())
26         front=0;
27     dizi[++rear]=eleman;
28 }
29
30 public T dequeue(){
31     if(isEmpty())
32         throw new RuntimeException("Queue is empty");
33     T donen=dizi[front++];
34     if(front>rear)
35         front=rear=-1;
36     return donen;
37 }
38 public int count() {
39     if(isEmpty())
40         return 0;
41     return rear-front+1;
42 }
43
44 public void print() {
45     for (int i = front; i <= rear ; i++) {
46         System.out.println(dizi[i]);
47     }
48 }
49 }
```

```
1 public class QueueMain {  
2     public static void main(String[] args) {  
3         MyQueue<Integer> q=new MyQueue<>();  
4         q.enqueue(5);  
5         q.enqueue(7);  
6         q.enqueue(4);  
7         q.dequeue();  
8         q.dequeue();  
9         q.dequeue();  
10        q.enqueue(6);  
11        q.print();  
12    }  
13 }
```

İşlem Önceliği

**İyi bir matematikçi iseniz
bu soruyu cevaplayın.**

$$4 \times 4 + 4 \times 4 + 4 - 4 \times 4 = ?$$

İnsanların %73'ü doğru cevabı bulamıyor.

Notasyonlar/Gösterimler

- **Infix:** İşlem(operator) ortada, değerler(operand) sağda ve solda
- **Prefix:** İşlem önde, değerler sonda
- **Postfix:** İşlem sonda, değerler önde

Postfix/Reverse Polish Notation

Example (Infix gösterimi)

$4 \times 4 + 4 \times 4 + 4 - 4 \times 4$

Postfix/Reverse Polish Notation

Example (Infix gösterimi)

$4 \times 4 + 4 \times 4 + 4 - 4 \times 4$

Example (Postfix gösterimi)

$4\ 4\ *\ 4\ 4\ * + 4 + 4\ 4\ * -$

Postfix/Reverse Polish Notation

Example (Infix gösterimi)

$3 * 2 + 4 - 7 / 5$

Postfix/Reverse Polish Notation

Example (Infix gösterimi)

$3 * 2 + 4 - 7 / 5$

Example (Postfix gösterimi)

$3 2 * 4 + 7 5 / -$

Postfix Değerlendirme Algoritması

İşlenenler(operands) için bir yığın(stack) oluştur

while *Girdi metninde değer olduğu sürece o değerini oku* **do**

if *okunan değer(o) sayı ise* **then**

 | Değeri yığita it

else

 | Yığıttan d1 ve d2 değerlerini çek

 | d2 o d1 yap

 | Sonucu yığita it

end

end

Yığıttaki tek değer sonucu verir

Algorithm 1: Postfix değerlendirme algoritması

Infix İfadeyi Postfix İfadeye Dönüştürme Algoritması

İşlemler için opstack yığını ve çıktı için output listesini oluştur

Girdi metnini sembollerine ayır

while *Girdi sembollerini soldan sağa oku(s)* **do**

if *Okunan sembol sayı ise* **then**

 | Sembolü çıktı listesine ekle

else if *Sembol parantez açma ise "("* **then**

 | Sembolü opstack yığına it

else if *Sembol parantez kapama ise ")"* **then**

 | opstack yığındaki "(" gelene kadar yığıttan elemanları çek ve çıktı listesine ekle

else

 | *// Sembol işlem ise(+,-,*,/)*

 | opstack yığında önceliği sembolden(s) daha büyük olan işlem olduğu sürece
 | yığıttan çek ve çıktı listesine ekle

 | Sembolü(s) opstack yığına it

end

end

opstack yığında kalan tüm işlemleri çek ve çıktı listesine ekle

Çıktı listesi postfix ifadeyi verir

Algorithm 2: Infix ifadeyi Postfix'e dönüştürme

