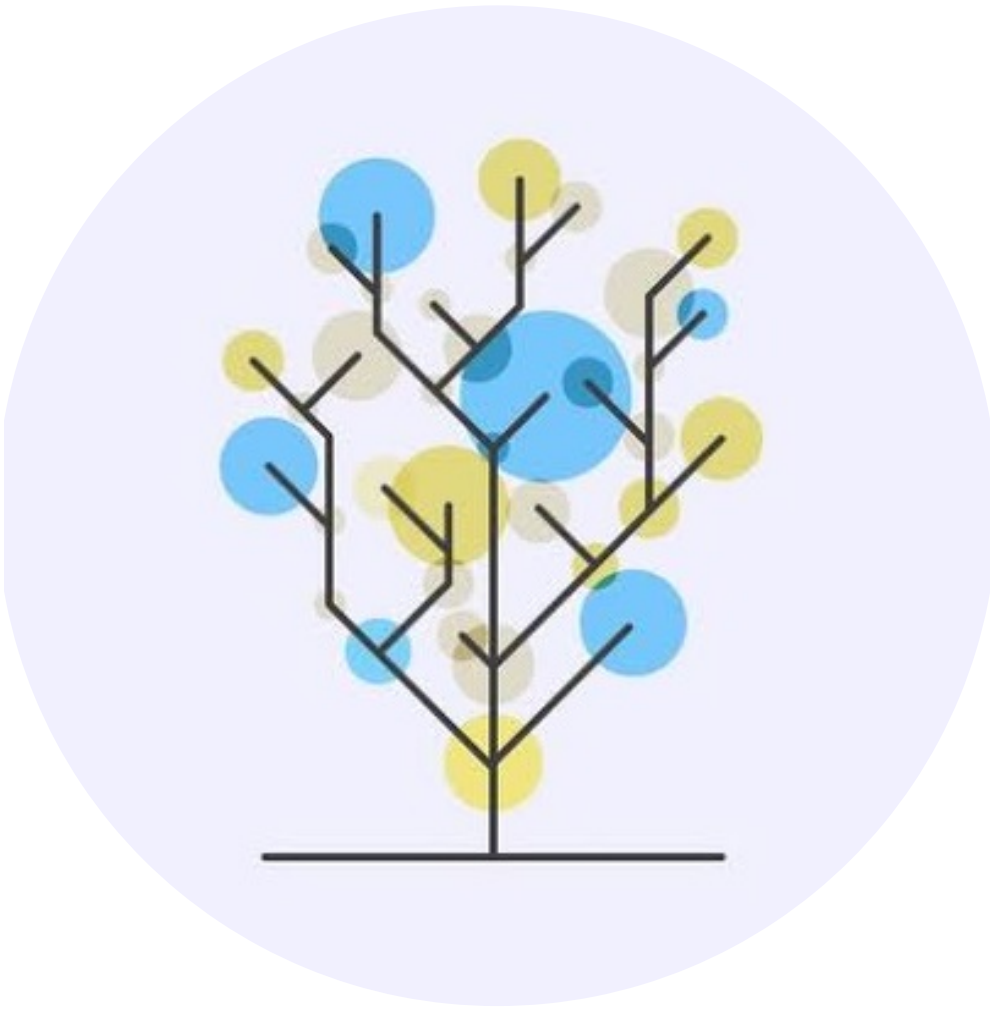


Stacks and Queues

04.01.2020

Data Structures & Algorithms

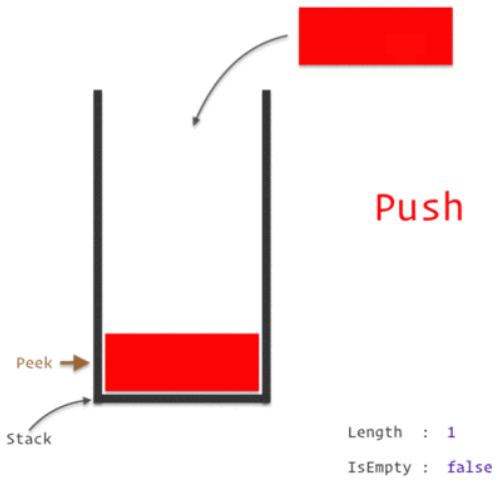
Intro

- Stacks ve Queues çok benzer oldukları için aynı anda öğreneceğiz.
- They are both called linear data structures allow us to traverse. That is go through elements sequentially one by one in which only one data element can be directly reached.
- Each of these structures differs by how it adds and removes items.

- Unlike arrays in stacks and queues there is no random access operation.
- Genelde stacks and queues uç elemenlarla yapılan işlemler için tercih edilir. (PUSH - POP - PEEK)

- We usually only can access the first and the last element! Bu biraz kısıtlayıcı değil mi neden kullanılıyoruz? Göreceğiz ki arrays veya linked lists kullanan stacks&queues oluşturacağız ve array ve linked lists'e göre daha az methodu olacak.
- Yani aslında amacımız zaten **LIMITING OPERATIONS THAT USER CAN DO**, böylece data structure'ı kontrol altında tutmuş oluruz.
- If you give someone all the tools in the world it's a lot harder for them to operate than if you just give them one or two so that they know exactly what they need to do.

Stacks

- Üstüste binen tabaklara benzetebiliriz, her tabak one piece of data'yı temsil eder.
- Gelen datalar üstüste biner.
- Addition ve Removal aynı uçtan yapılır.
- **Last In First Out - LIFO**
- Yani sadece son girene ulaşabiliyoruz, aynı en üstteki tabağa ulaşmak gibi.
- 

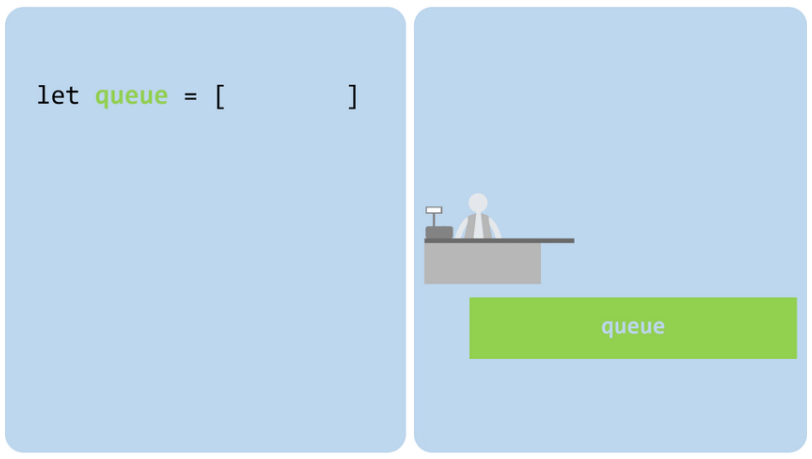
- STACKS ARE FUNDAMENTALLY IMPORTANT, AS THEY CAN BE USED TO **REVERSE THE ORDER OF ITEMS**
- THE ORDER OF INSERTION IS THE REVERSE OF THE ORDER OF REMOVAL.

- Mesela Web Browser'lar bu yapıyı kullanır, ziyaret edilen URLs bir stack içine kaydedilir böylece geri tuşu ile bir önceki websitesine geri dönebiliriz.Bu örnekte TOP şu anda ziyaret edilen sayfa iken, BASE ise ilk bakılan sayfa oluyor.

- **POP : O(1)** --> Remove the top plate(last item).
  PUSH : O(1) --> Add a plate.
  PEEK : O(1) --> View the top plate(last item).

  LOOKUP : O(n) --> BU GENELDE KULLANILMAZ AMA KULLANILIRSA O(n) olacağını bil. Tek tek dolanmamız gerekecek.

Queues

- Stacks'i tabaklara benzetmiştik
- Queues'da bir sıra gibi düşünülebilir. Sıraya ilk giren ilk çıkar. **First in First Out - FIFO**
- 

- **ENQUEUE : O(1)** --> Add to the queue. Sıraya bir kişi daha eklenir.
  DEQUEUE : O(1) --> Remove person from line (En baştaki)
  PEEK : O(1) --> View the first person in the line.

  LOOKUP : O(n) --> Genelde kullanılmaz.

- **Why would you not use array to build a queue?**
  - Çünkü çok verimsiz olur. First item remove edilirse, arrayin tüm elemanları shift olacak! Direkt O(n)!

Exercise

- Stacks browser history için kullanılabilir.
- Stacks'i ARRAYS veya LINKED LISTS ile build edebiliriz.

- **SORU:** Bir stacks class'ı oluşturunca datayı array içinde tutmamınızın avantajı ne olur linked list içinde tutmamızın avantajı ne olur
  - Bana farketmez gibi göründü son elemanla işlem yaptığımız için ikisi de O(1) olur.
  - İKİSİ DE İYİ ÇALIŞIR. BURADA İKİSİNİN ÖZELLİKLERİNİ DE DÜŞÜNÜP UYGULAMA BAZLI BİRİNİ SEÇEBİLİRİZ.MESELA ARRAY: ARRAY ELEMANLARI HAFIZADA YANYANA OLDUGU İÇİN (CASH LOCALITY) DAHA HIZLI ULAŞIM. BUNA KARŞI LINKED LIST ELEMANLARI HAFIZAYA DAGILMIŞ DURUMDA VE HER BIR ELEMAN EKSTRA MEMORY TUTAR POINTERS. ANCAK LINKED LISTS'E İSTEDİĞİMİZ KADAR ELEMAN EKLEYEBİLİRİZ AMA ARRAY ICIN LIMITINE ULASINCA BASKA YERE TASINIYOR
- **SORU**: Diyelimki bir rezervasyon uygulaması yapıyoruz ilk gelen ilk işleme alınacak queues kullanacağız. Array mi Linked List mi daha mantıklı?
  - ARRAY MANTIKSIZ ÇÜNKÜ QUEUE İÇİN REMOVE İŞLEMİ 1. ELEMANA UYGULANACAK, ARRAY KULLANIRSAK KALAN ELEMANLARIN HEPSİ SHİFT OLACAK BU VERİMSİZ

Implement Stack by using a Linked List

```
//Implement a stack by linked list:

class Node {
  constructor(value){
    this.value = value;
    this.next = null;
  }
}

class Stack {
  constructor(){
    this.top = null;
    this.bottom = null;
    this.length = 0;
  }
  peek() {
    return this.top;
  }
  push(value){
    const newNode = new Node(value);
    if (this.length === 0) {
      this.top = newNode;
      this.bottom = newNode;
    } else {
      const holdingPointer = this.top;
      this.top = newNode;
      this.top.next = holdingPointer;
    }
    this.length++;
    return this;
  }
  pop(){
    if (!this.top) {
      return null;
    }
    if (this.top === this.bottom) {
      this.bottom = null;
    }
    const holdingPointer = this.top; //Bu top'u tutmazsak kendi kendine silinir. Silinse de olur
göstermek için yaptı.
    this.top = this.top.next;
    this.length--;
    return this;
  }
  //isEmpty
}

const myStack = new Stack();
myStack.peek();
myStack.push('google');
myStack.push('udemy');
myStack.push('discord');
myStack.peek();
myStack.pop();
myStack.pop();
myStack.pop();
```

```
//Discord
//Udemy
//google
```

Implement Stack by using an Array

```
// Now Implement Stack by using an Array:

// Bu çok daha kolay olacak çünkü zaten push pop gibi methodlar arrays için tanımlanmış

class Stack {
  constructor(){
    this.array = [];
  }
  peek() {
    return this.array[this.array.length-1];
  }
  push(value){
    this.array.push(value);
    return this;
  }
  pop(){
    this.array.pop();
    return this;
  }
}

const myStack = new Stack();
myStack.peek();
myStack.push('google');
myStack.push('udemy');
myStack.push('discord');
myStack.peek();
myStack.pop();
myStack.pop();
myStack.pop();


//Discord
//Udemy
//google
```

Implement Queue by using a Linked List

```
class Node {
  constructor(value) {
    this.value = value;
    this.next = null;
  }
}

class Queue {
  constructor(){
    this.first = null;
    this.last = null;
    this.length = 0;
  }
  peek() {
    return this.first;
  }
  enqueue(value){
    const newNode = new Node();

  if (this.length === 0){
    this.first = newNode;
    this.last = newNode;
  } else{
    this.last.next = newNode;
    this.last = newNode;
  }

  this.length++;
  return this;
}

dequeue(){

  if(!this.first){
    return null;
  }

  if(this.length==1){
    this.last = null;
    }

    this.first = this.first.next; //Böylece eski first'i point eden bir pointer kalmamış oldu ve
bu kendi kendine silinecek.
    this.length--;
    return this;


  }
  //isEmpty;
}

const myQueue = new Queue();
myQueue.peek();
myQueue.enqueue('Joy');
myQueue.enqueue('Matt');
myQueue.enqueue('Pavel');
myQueue.enqueue('Samir');
```

```
//Joy
//Matt
//Pavel
//Samir
```

Implement Queues by using a Stack

```
class CrazyQueue {
  constructor() {
    this.first = [];
    this.last = [];
  }

  enqueue(value) {
    const length = this.first.length;
    for (let i = 0; i < length; i++) {
      this.last.push(this.first.pop());
    }
    this.last.push(value);
    return this;
  }

  dequeue() {
    const length = this.last.length;
    for (let i = 0; i < length; i++) {
      this.first.push(this.last.pop());
    }
    this.first.pop();
    return this;
  }
  peek() {
    if (this.last.length > 0) {
      return this.last[0];
    }
    return this.first[this.first.length - 1];
  }
}

const myQueue = new CrazyQueue();
myQueue.peek();
myQueue.enqueue('Joy');
myQueue.enqueue('Matt');
myQueue.enqueue('Pavel');
myQueue.peek();
myQueue.dequeue();
myQueue.dequeue();
myQueue.peek();
```

Review

- They are good for
  - Fast operations (removing or inserting but at the **end** or **beginning**)
  - Fast peek (access to first or last item)
  - Ordered
- They are bad for
  - Lookups
  - Searchs

    - We don't use queues and stacks for lookups and Searchs
- They are usefull for restricting our use of datastructure.

More blogs