Stacks and Queues

What is a Stack?

Stores a set of elements in a particular order

Accessed in Last-In-First-Out (LIFO) fashion

Real life examples:

- Pile of books
- PEZ dispenser
- Cup trays in cafeteria

CS examples: program execution stack, parsing/evaluating expressions

Stack Abstract Data Type

push(*o*): insert *o* on top of stack

pop( ): remove object from top of stack

top( ): look at object on top of stack (but don't remove)

size( ): number of objects on the stack

isEmpty( ): does (size = = 0)?

Java Interface for Stack ADT

```
public interface Stack {
  public int size();
  public boolean isEmpty();
  public Object top() throws
      StackEmptyException;
  public void push (Object element);
  public Object pop() throws
      StackEmptyException;
}
```


Array-based Stack Implementation

Allocate array of some size

- Maximum # of elements in stack

Bottom stack element stored at index 0

first index tells which element is the top

increment *first* when element pushed, decrement when pop'd

Array-based Implementation

```
public class ArrayStack implements Stack {
  private Object[] S;
  private int topIndex = -1;
  public void push(Object obj) throws StackFull {
      if (size() == S.length)
              throw new StackFull("full");
      S[++topIndex] = obj; }
  public Object pop() throws StackEmpty {
      if (isEmpty())
               throw new StackEmpty("empty");
      Object elem = S[topIndex];
          S[topIndex--] = null;
      return elem; }
}
```

Analysis

Each operation is $O(1)$ running time

- Independent of number of items in stack
- push, pop, top, size, isEmpty

Space can be $O(n)$ or may be much more

- depends if n is known at initialization time

Linked List Stack Implementation

Benefits

- Avoids maximum stack size restriction
- Only allocates memory for stack elements actually used

How

- Allocate a node for each stack element
- Nodes are chained together by reference to next node in the chain

Linked List Node

```
public class Node {
  private Object element;
  private Node next;
  public Node(Object e, Node n) {
      element = e; next = n; }
  …
  }
```


Linked List Stack Implementation

```
public class LinkedStack implements Stack {
  private Node top = null;
  private int size = 0;
  public void push(Object elem) {
      Node v = new Node();
      v.setElement(elem);
      v.setNext(top);
      top = v;
      size++; }
  public Object pop() throws StackEmpty {
      if (isEmpty()) throw new
             StackEmpty("empty");
      Object temp = top.getElement();
      top = top.getNext();
      size --;
      return temp; }
}
```


Analysis

All stack functions still $O(1)$

- push, pop, top, size, isEmpty

What is a Queue

Stores a set of elements in a particular order

Accessed in First-In-First-Out (FIFO) fashion

Real life examples:

- Waiting in line at cafeteria
- Waiting on hold for technical support

CS Example: Buffered I/O

Queue ADT

enqueue(*o*): Insert object *o* at *rear* of queue

dequeue( ): remove object from *front* of queue

size( ): number of elements

isEmpty( ): size == 0?

front( ): look at object at front of queue

Queue Interface in Java

```
public interface Queue {
  public int size();
  public boolean isEmpty();
  public Object front() throws QueueEmpty;
  public void enqueue (Object element);
  public Object dequeue() throws QueueEmpty;
}
```


Array-based Queue Implementation

Array of fixed size

Index array element for front and rear of queue

Indices “wrap around” when they cross end of array

Array Queue Implementation

```
public class ArrayQueue implements Queue {
  private Object[] Q;
  private int size=0;
  private int front=0, rear = 0;
  public void enqueue(Object o) {
      if (size( ) == Q.length) throw
          new QueueFull("full");
      Q[rear] = o;
      rear = (rear + 1) % Q.length;
      size++;
  }
```


List Queue Implementation

Head and tail node references for front and rear of queue

Insert at tail, remove from head

- Remove from tail too slow for singly linked list
  - —Updating tail reference with new tail takes full traversal
- So use tail of list for rear of queue

List Queue Implementation

```
public class ListQueue
 implements Queue {
 private Node head = null;
 private Node tail = null;
 private int size = 0;
  . . .
}
```

List Queue

```
public void enqueue(Object obj) {
  Node node = new Node(obj, null);
  if (size == 0)
      head = node;
  else
      tail.setNext(node);
  tail = node;
  size++;
}
public Object dequeue() {
  Object obj = head.getElement();
  head = head.getNext();
  size--;
  if (size == 0)
      tail = null
  return obj;
}
```


Analysis

All queue operations are $O(1)$

- size( ), isEmpty( )
- enqueue( ), dequeue( ), front( )

Double-ended Queue

Sometimes called “deque” (děk)

Similar to stack and queue

- Allows insertion and removal at both ends of the queue
- Stack or queue is easily implemented using a deque

Deque ADT

insertFirst(*e*) : insert element at front
insertLast(*e*) : insert element at rear
removeFirst( ) : remove first element
removeLast( ) : remove element at rear
first( ) : examine first element
last( ) : examine last element
size(*e*) : number of elements in deque
isEmpty( ) : size == 0?

Doubly Linked List

Singly linked list inefficient for removing from tail

Has prev reference as well as next reference

Can use *sentinel* nodes to reduce the special cases

- node has no element, just next or prev reference

Deque Implementation

```
public class MyDeque implements Deque {
 DLNode header, trailer;
 int size;
 public MyDeque() {
    header = new DLNode();
    trailer = new DLNode();
    header.setNext(trailer);
    trailer.setPrev(header);
    size = 0;
}
```

Deque Implementation

```
public void insertFirst(Object o) {
  DLNode second = header.getNext();
  DLNode first =
     new DLNode(o,header, second);
  second.setPrev(first);
  header.setNext(first);
  size++;
}
```

Could be null if no sentinels
(trailer could also be null)

Deque (no sentinels)

```
public void insertFirst(Object o) {
  DLNode second = header;
  DLNode first = new DLNode(o, null, second);
  header = first;
  if (second == null)
     trailer = first;
  else
     second.setPrev(first);
  size++;
}
```


Deque Analysis

All operations still *O*(1)

Doubly linked list

- **nodes slightly larger**
- **more references to keep up to date**