Stacks and Queues

CSE 373
Data Structures
Lecture 6

Stacks

• A list for which Insert and Delete are allowed only at one end of the list (the top)
  › the implementation defines which end is the "top"
  › LIFO – Last in, First out
• Push: Insert element at top
• Pop: Remove and return top element (aka TopAndPop)
• isEmpty: test for emptiness

Important Applications of Stacks

• Parsing phase in compilers
• Call stack in run time systems
  › When a function (method, procedure) is called the work area (local variables, copies of parameters, return location in code) for the new function is pushed on to the stack. When the function returns the stack is popped.

Two Basic Implementations of Stacks

• Linked List
  › Push is InsertFront
  › Pop is DeleteFront
  › isEmpty is test for null
• Array
  › The k items in the stack are the first k items in the array.

Linked List Implementation
Array Implementation

- Stack of blobs

```
A
   bottom   top
1   2   3   4   5   6   7   8   9  10  11  12
```

holder = blob pointer array
size = number in stack
maxsize = max size of stack

Push and Pop

- IsEmpty(A : blob stack pointer) : boolean {
  return A.size = 0
}
- IsFull(A : blob stack pointer) : boolean {
  return A.size = A.maxsize;
}
- Pop(A : blob stack pointer) : blob pointer {
  // Precondition: A is not empty
  // A.size = A.size - 1;
  return A.holder[A.size + 1];
}
- Push(A : blob stack pointer, p : blob pointer) {
  // Precondition: A is not full
  A.size = A.size + 1;
  A.holder[A.size] = p
}

Linked Lists vs Array

- Linked list implementation
  - flexible – size of stack can be anything
  - constant time per operation
  - Call to memory allocator can be costly
- Array Implementation
  - Memory preallocated
  - constant time per operation.
  - Not all allocated memory is used
  - Overflow possible - Resizing can be used but
  but some ops will be more than constant time.

Queue

- Insert at one end of List, remove at the
  other end
- Queues are “FIFO” – first in, first out
- Primary operations are Enqueue and Dequeue
- A queue ensures “fairness”
  - customers waiting on a customer hotline
  - processes waiting to run on the CPU

Queue ADT

- Operations:
  - Enqueue - add an entry at the end of
    the queue
  - Dequeue - remove the entry from the front
    of the queue
  - IsEmpty
  - IsFull may be needed

Applications of Queues

- File servers: Users needing access to
  their files on a shared file server
  machine are given access on a FIFO
  basis
- Printer Queue: Jobs submitted to a
  printer are printed in order of arrival
- Phone calls made to customer service
  hotlines are usually placed in a queue
**Pointer Implementation**

- The figure shows a simple linked list implementation of a queue. The queue consists of nodes, each containing a value and a pointer to the next node.
- The front and rear pointers are used to keep track of the first and last elements in the queue.

**List Implementation**

- The code snippet demonstrates the implementation of a queue using pointers.
- The `IsEmpty` function checks if the queue is empty.
- The `Dequeue` function removes the front element from the queue.
- The `Enqueue` function adds a new element to the end of the queue.

**Array Implementation**

- This section explains the circular array implementation of a stack.
- The array is used to store the stack elements, and indices are used to manipulate the stack.
- The `rear` pointer wraps around when it reaches the end of the array.

**Wrap Around**

- The diagram illustrates how the `rear` pointer wraps around in a circular array.
- The equation `rear = (front + size) mod maxsize` shows how the pointer wraps when it reaches the end of the array.

**Enqueue**

- The enqueue operation is demonstrated with an example stack.
- The new element is added to the end of the stack, and the `rear` pointer is updated accordingly.
Enqueue

Enqueue(Q : blob queue pointer, p : blob pointer) : { 
// precondition : queue is not full 
Q.holder[(Q.front + Q.size) mod Q.maxsize] := p; 
Q.size := Q.size + 1; 
}

Constant time!

Dequeue

Dequeue(Q : blob queue pointer) : blob pointer { 
// precondition : queue is not empty 
Q.front := (Q.front + 1) mod Q.maxsize; 
Q.size := Q.size - 1; 
return p; 
}

Try Dequeue

• Class Participation
• Define the circular array implementation of Dequeue

Solution to Dequeue