Lectures: Data Structures

Programming Systems and Tools

State University of New York at Binghamton

April 11, 2024

BINGHAMTON UNIVERSITY | THOMAS J. WATSON COLLEGE OF ENGINEERING AND APPLIED SCIENCE

Stacks and Queues

Abstract Data Types

- Data Structures (DS) are the implementation that manage data
- Abstract Data Types (ADT) describe behavior, not implementation
  - A DS is an implementation of an ADT
  - DS should have a standardized interface, but the underlying implementation can vary greatly

Stacks

Stacks are a sequential collection:

Sequential Order of element insertion matters

Collection Contains items of the same type, forming a grouping

- The purpose of a Stack ADT is to describe how the data is accessed, not how the data is stored!
- Two popular Stack DS implementations are linked list and array
- The Stack DS implementations provide the ADT's sequential access control through encapsulation

Stack Access Control

- Stacks are **Last In, First Out** (LIFO)
- Items are removed in the reverse order of their insertion.
- Think of stacks as being like a pile of plates: to get to a lower plate, we have to lift the plates above it.
- Stacks limit the user to accessing only the newest element.

Examples of Stacks

What are some examples of stacks in computer science?

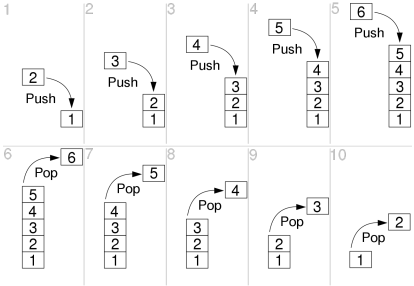
Examples of Stacks

What are some examples of stacks in computer science?

- Function call stack
- Exception handling/unwinding
- Undo/Redo commands
- Syntax checking - are brackets properly matched?
- Prefix notation calculators (“RPN”)

Stack Update Operations

- Push
  - Inserting an item onto the stack
  - Imagine pushing down on everything already on the stack, one place farther from the top
- Pop
  - Removing from the stack
  - Each element remaining on the stack moves one place closer to the top
  - To access an element below, you must pop every element above it
  - `top()` or `peek()` get the top item like `pop()` but without removing it from the stack



Stack Implementations

Array Based

- ‘top’ is index of the most recently appended item
- Top of the stack is at the higher address than the bottom

Pros of Array Based

- Easier memory management

Cons of Array Based

- Must have a static, maximum size or a dynamic vector storage

Stack Implementations

Linked List Based

- 'top' is a pointer to the head of the list
- Stack grows at the head of the list

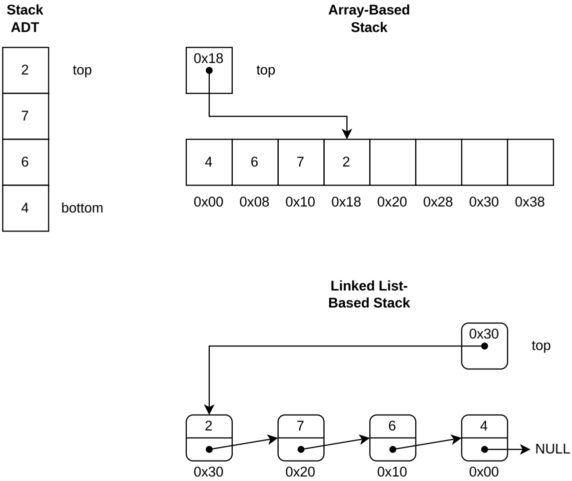
Pros of Linked List Based

- Internal storage is dynamically allocated $\rightarrow$ always right-sized

Cons of Linked List Based

- Push and pop have increased memory management

Stack Implementations



Stack Interface

```c
#ifndef DATA_T
#define DATA_T
typedef uint64_t Data;
#endif // DATA_T

#ifndef STACK_H
#define STACK_H

typedef struct Node {
  Data value;
  struct Node *next;
} Node;

typedef struct Stack {
  Node *top;
  size_t size;
} Stack;
```

Stack Interface

```
// Not shown: new, delete, print methods
void clear(Stack *s);
size_t size(Stack *l);
Node *top(Stack *s);
Node *pop(Stack *s);
Node *push(Stack *s, Data d);
void traverse(Stack *s, int (*func)(const void *));
```

Queues

Like stacks, queues are also a sequential collection:

Sequential
: Order of element insertion matters

Collection
: Contains items of the same type, forming a grouping

- Like stacks, queues provide restricted access control through encapsulation
- Unlike stacks, queues are First In First Out (FIFO)

Queue Access Control

- Queues are **First In, First Out** (FIFO)
- Items are removed in the order of their insertion.
- Think of queues as being like a line at the airport: each element must wait until all other elements that arrived first are processed.
- Queues limit the user to accessing only the oldest element.

Examples of Queues

What are some examples of queues in computer science?

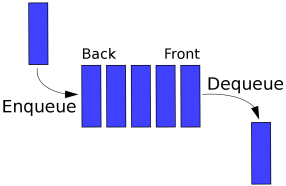
Examples of Queues

What are some examples of queues in computer science?

- I/O buffers
- Operating system pipes
- Multithreading synchronization primitives
- First Come, First Serve (FCFS) process scheduling
- Cryptographic block and stream ciphers

Queue Update Operations

- Enqueue
  - Inserting an item into the queue
  - Imagine adding something to the end of the line. No other items' position needs to change as long as the queue grows to the back.
- Dequeue
  - Removing an item from the queue
  - Each element remaining in the queue moves one place closer to the front
  - To access an element behind, you must dequeue every element inserted before it
  - `front()` or `peek()` get the front item like `dequeue()` but without removing it from the queue



Queue Implementations

Array Based

- ‘head’ and ‘tail’ are pointers to the front and end of the queue in the array
- Head of the queue is at a lower address than the tail of the queue

Pros of Array Based

- Easier memory management

Cons of Array Based

- Requires array to have a static, maximum size or a dynamic vector

Queue Implementations

Linked List Based

- 'head' is a pointer to the head of the list, 'tail' is a pointer to the end of the list
- Queue grows at the tail of the list

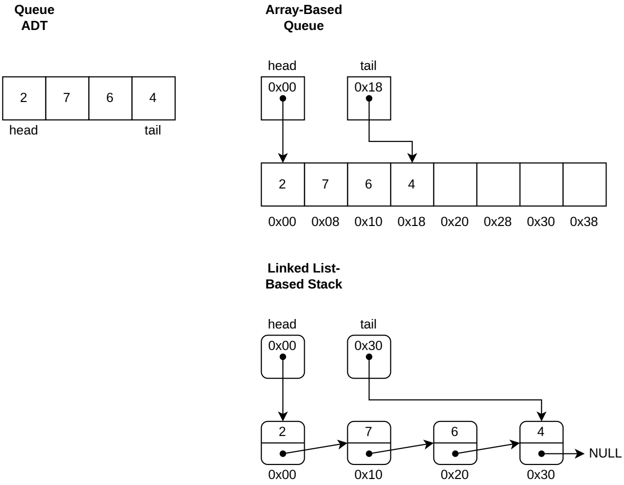
Pros of Linked List Based

- Internal storage is dynamically allocated → always right-sized

Cons of Linked List Based

- Enqueue/Dequeue have increased complexity

Queue Implementations



Queue Interface

```c
#ifndef DATA_T
#define DATA_T
typedef uint64_t Data;
#endif // DATA_T

#ifndef QUEUE_H
#define QUEUE_H

typedef struct Node {
  Data value;
  struct Node *next;
} Node;

typedef struct Queue {
  Node *head, *tail;
  size_t size;
} Queue;
```

Queue Interface

```c
/* Not shown: new, delete, print methods */
Node *back(Queue *q);
void clear(Queue *q);
Node *dequeue(Queue *q);
Node *enqueue(Queue *q, Data d);
Node front(Queue *q);
size_t size(Queue *q);
void traverse(Queue *q, int (*func)(const void *));
```

Circular Buffer

A specialized implementation of queues, using a single, fixed-size buffer with wraparound addressing as though the buffer was connected end-to-end. Like queues, circular buffers are also a sequential collection:

Sequential: Order of element insertion matters

Collection: Contains items of the same type, forming a grouping

- Like queues, circular buffers provide restricted access control through encapsulation
- Like queues, circular buffers are well-suited to be used for First In First Out (FIFO) access control, except they have the advantage of not needing any memory resizing or copying (all operations are constant time).

Circular Buffer Access Control

- Circular buffers are **First In, First Out** (FIFO)
- Items are removed in the order of their insertion.
- Think of circular buffers being like clocks, where the hands point to the item in storage to be read, or the location in storage to write the next item.

Examples of Circular Buffers

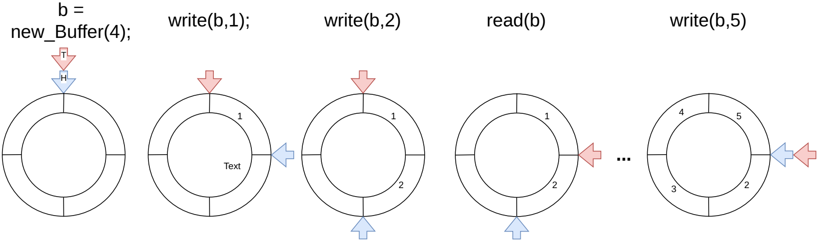
What are some examples of circular buffers in computer science?

Examples of Circular Buffers

What are some examples of circular buffers in computer science?

- Keyboard input
- Producer-consumer pattern

Circular Buffer Implementation



Write:

1. buffer[tail] $\leftarrow$value
2. tail $\leftarrow$ $tail + 1$ mod $capacity$

Remove:

1. temp $\leftarrow$ buffer[tail]
2. tail $\leftarrow$ $tail - 1$ mod $capacity$
3. return temp

Read:

1. temp $\leftarrow$ buffer[head]
2. head $\leftarrow$ $head + 1$ mod $capacity$
3. return temp

Circular Buffer Interface

```c
#include <stdint.h>
#include <stddef.h>

typedef uint64_t Data;
typedef struct Buffer {
  size_t *head,*tail;
  Data* storage;
  size_t size;
}

// Not shown:
// new, copy, del methods for Buffer
void clear(Buffer *b);
void write(Buffer *b, Data d);
Data read(Buffer *b);
```

Double-Ended Queue

A **double-ended queue** or **deque** (pronounced *deck*) is a generalization of the queue plus the stack. Deques are still sequential collections:

Sequential
: Insertion direction and timing creates an ordering

Collection
: Contains items of the same type, forming a grouping

- Deques provide restricted access control through encapsulation
- Insertion and removal can happen at either side of the storage, but not in the middle of the storage.

Deque Implementations

- Deques can be implemented with different types of storage:
  - head-tail linked list (most common)
  - dynamic array (sometimes implemented as a circular array when maximum storage size is known)
- Deques can be implemented with different access restrictions:
  - Insertion and deletion at both ends (most common)
  - Input restricted: deletion from both ends, but insertion at one end only
  - Output restricted: insertion at both ends, but deletion at one end only

Examples of Deques

What are some examples of deques in computer science?

Examples of Deques

What are some examples of deques in computer science?

- Scheduling with “work stealing” for multiprocessor computers and parallel programming (A-steal algorithm)
- “Sliding window” problems
- Browser history with bounded size (LIFO) plus forward and back buttons (FIFO)

Deque Operation

The basic operations of the deque are enqueue and dequeue at either end:

Enqueue:

- Insert element at back:
  - inject (textbooks)
  - push back (C-like languages)
  - ...or append, snoc
- Insert element at front:
  - push (textbooks)
  - push front (C-like languages)
  - ...or prepend, unshift, cons

Dequeue:

- Remove element at back:
  - eject (textbooks)
  - pop back (C-like languages)
  - ...or delete
- Remove element at front:
  - pop (textbooks)
  - pop front (C-like languages)
  - ...or shift
- Deque implementations also sometimes offer a "peek" that returns the value without dequeing it.

Deque Operation