CSCI 136:
Data Structures
and
Advanced Programming

Lecture 18

Linear structures

Instructor: Dan Barowy

Williams

Topics

- Stack data structure
- Queue ADT
- Queue data structure

Your to-dos

1. Lab 6 (partner lab), **due Tuesday 4/12 by 10pm**.
   a. (Anti-)partner form if you have feelings.
2. Read **before Fri**: Bailey, Ch 8-8.3.

Announcements

Colloquium on Friday.

Friday, April 8 @ 2:35pm

Wege Hall – TCL 123

Perception and Context in Data Visualization

Jordan Crouser, Smith College

Visual analytics is the science of combining interactive visual interfaces and information visualization techniques with automatic algorithms to support analytical reasoning through human-computer interaction. People use visual analytics tools and techniques to synthesize information and derive insight from massive, dynamic, ambiguous, and often conflicting data… and we exploit all kinds of perceptual tricks to do it! In this talk, we'll explore concepts in decision-making, human perception, and color theory as they apply to data-driven communication. Whether you're an aspiring data scientist or you're just curious about the mechanics of how data visualization works under the hood, stop by and take your pre-attentive processing for a spin.

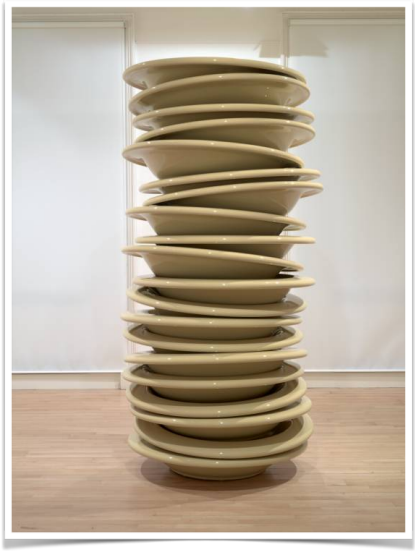
Announcements

Please **consider being a TA** next semester (especially for this class!)

Applications **due Friday, April 22**.

https://csci.williams.edu/tatutor-application/

Stack ADT



Stack ADT

A **stack** is an **abstract data type** that stores a collection of **any type of element**. A stack **restricts which elements are accessible**: elements may only be added and removed from the "**top**" of the collection. The "**push**" operation places an element onto the top of the stack while a "**pop**" operation removes an element from the top.

Stack implementations

StackArray

A **StackArray** is a stack implemented using an **array** for element storage.

Pros: **push** and **pop** are **O(1)** operations.

Cons: data structure has a maximum **capacity**.

Stack implementations

StackVector

A **StackVector** is a stack implemented using a **Vector** for element storage.

Pros: **push** and **pop** are amortized **O(1)** operations. There is no maximum capacity.

Cons: Most of the time, ops take **O(1)** time, but occasionally--when the underlying array needs to grow--an **O(n)** cost is incurred. This may be fine for most applications, but if the application cannot tolerate wide variation in time, this is a bad choice.

Also, unless the underlying array is completely full, Vectors **waste some space**.

Stack implementations

StackList

A **StackList** is a stack implemented using a **List** (usu. **SLL**) for element storage.

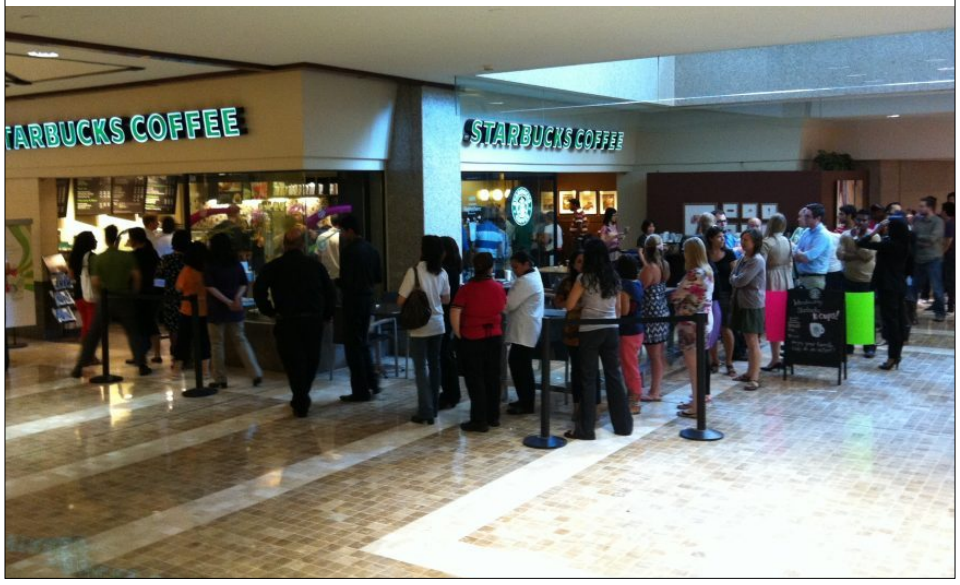
Pros: **push** and **pop** are **O(1)** operations. There is no maximum capacity, and no wasted space. **push** and **pop** costs are predictable (always the same), unlike **StackVector**.

Cons: because of the way computer hardware is implemented, a **StackList**'s constant-time cost is likely to be much higher than a **StackVector**'s. So a **StackList**'s performance may be **more predictable** than a **StackVector**, but it will likely be **slower on average**.

Queue ADT

A **queue** is an **abstract data type** that stores a collection of **any type of element**. A queue **restricts which elements are accessible**: elements may only be added to the "**end**" of the collection and elements may only be removed from the "**front**" of a collection. The "**enqueue**" operation places an element at the end of a queue while a "**dequeue**" operation removes an element from the front.

Queue ADT



Queue ADT

Also sometimes referred to as a **FIFO**: **"first in, first out."**

(a stack would be an annoying way to process a line at Starbucks!)

Frequently used as a **buffer** to hold work **to do later**.

We also frequently include a "**peek**" operation that lets us look at an element on the top of a queue without removing it, and "**size**" and "**isEmpty**" operations that let us check how many elements are stored and whether a queue stores zero elements, respectively.

Queue implementations

QueueArray

A **QueueArray** is a queue implemented using an **array** for element storage.

Pros: **enqueue** and **dequeue** are **O(1)** operations.

Cons: data structure has a maximum **capacity**.

Queue implementations

QueueVector

A **QueueVector** is a queue implemented using a **Vector** for element storage.

Pros: **enqueue** and **dequeue** are amortized **O(1)** operations. There is no maximum capacity.

Cons: Most of the time, they take **O(1)** time, but occasionally--when the underlying array needs to grow--an **O(n)** cost is incurred. This may be fine for most applications, but if the application cannot tolerate wide variation in time, this is a bad choice. Also, unless the underlying array is completely full, **Vectors waste some space**.

Queue implementations

QueueList

A **QueueList** is a queue implemented using a **List** (usu. **DLL** or **CL**) for element storage.

Pros: enqueue and dequeue are **O(1)** operations. There is no maximum capacity. **enqueue** and **dequeue** costs are predictable (always the same), unlike **QueueVector**.

Cons: because of the way computer hardware is implemented, a **QueueList**'s constant-time cost is likely to be much higher than a **QueueVector**'s. So a **QueueList**'s performance may be more predictable than a **QueueVector**, but it will likely be slower on average.

Recap & Next Class

Today:

Queue ADT

Data structure choices for linear structures

Next class:

Search

Iterators