EECS 211, Fall 2014

Stacks and Queues

William Hendrix

Lecture 27

Today

- Abstract data types

- Stacks and queues
    - Operations
    - Implementation

- Deques

- Minilab

Stacks and queues

- Abstract data types: may be implemented with different underlying representation
  - E.g., LinkedList or array
- Queue
  - Like a line at the grocery store
    - People (elements) queue up and are served in the order they arrived
    - "First in, first out" (FIFO)
  - Two main operations: `enqueue` and `dequeue` (`push` and `pop`)
    - `enqueue/push` adds elements to the queue
    - `dequeue/pop` removes elements in the order they were added
  - Optional functions: `size`, `isEmpty`, `peek`
- Stack
  - Like a stack of plates on the counter
    - Plates (elements) can only be added to or removed from top of stack
    - "Last in, first out" (LIFO)
  - Operations: `push` and `pop`

The call stack

- Function call stack is a stack

- Every time you call a function, return address and another "stack frame" are pushed onto stack
  - Stack frames store local variables for functions

- On return, stack frame is "popped" and program returns to return address
  - Stack behavior: returns to calling function

- **Example**

`main()`	`fact(3)`	`fact(2)`	`fact(1)`	`fact(0)`

- All recursive functions can be rewritten as a loop with a stack

Stack and queue implementation

- LinkedList implementation is trivial
  - Stack: add and delete from end
  - Queue: add to end, delete from beginning
  - Slower than arrays, no case where we insert in middle of list
- Array-based stack
  - Add and remove elements from end
  - When adding to full array, reallocate (double size) and copy
  - Data members: array, size, capacity
- Array-based queue
  - Add to end, remove from beginning
  - Instead of shifting elements around when removing, we just change where the queue "starts"
  - Data members: array, head, tail, capacity
  - When tail of queue reaches the end, we can reuse space at the beginning vacated by head ("circular" array)
  - Queue is full when tail reaches head
    - Reallocate and copy

Deques

- “Double-ended queue”
  - Can serve as stack *or* a queue
- Syntax

```
#include <deque>
//...
deque<int> deq;
```

- Can store any data type (like `vector`)
- Main operations
  - `void push_back(<type>)`
  - `<type> back(), <type> front()`
  - `void pop_back(), void pop_front()`
- Other operations
  - `void push_front(<type>)`
  - `int size()`
  - `operator[], at(index)`
  - `void clear()`
  - `bool empty()`

Queue and stack applications

- Stacks and queues are used frequently throughout computer science

- Queues are often used as to coordinate requests for resources
  - E.g., buffering output to files, scheduling files for printer

- Stacks are used for many different purposes
  - Matching parentheses in algebraic expressions
    - Push `'('`, pop at `')'`
    - If stack runs out of `'('` or has extra at end, not correct
  - Reversing a string
    - Push all characters, pop them all off

Stack application: backtracking search

- Imagine trying to solve a maze
  - You continue until you reach a split in the path
  - You try one path and continue until you win or reach a dead end
  - When you hit a dead end, back up to last decision point and try another direction
  - Repeat until you win or exhaust options
- When making a move, push it on a stack
- When you reach a dead end, pop from stack to reverse steps
- Continue backwards until a different forward move is possible
  - Always try to make moves in a consistent order

- Backtracking is a powerful problem-solving strategy
- Works for any problem where you can:
  - represent your current state/location
  - generate new states based on your current one
  - recognize your goal
- May not succeed if you can't generate all possible "moves"

Minilab

- Use a deque to implement the `solve()` member function of the provided Maze class.
  - Use `push_back()`, `back()`, and `pop_back()`

- Relevant functions
  - void getMoves(bool& up, bool& right, bool& left, bool& down);
    - Returns whether you can move up, right, left, or down from current position
  - bool move(direction_t dir);
    - Moves UP, DOWN, LEFT, or RIGHT
  - bool move_back(direction_t dir)
    - "Undoes" a move
  - bool isSolved()
    - Returns whether you've found the Maze exit

Tonight

- **Recommended reading:** Sections 13.2 and 13.3
- **Recommended exercise:** finish minilab