More Lists ADTs

Stacks and Queue
Stacks

- A stack is a list with the restriction that insertions and deletions can only be done at the end of the list (called the top).
- Called a *push* to insert, a *pop* to delete.
- Also called “Last In First Out” (LIFO).
Imagine a stack of plates on a popup dispenser at a cafeteria.
Stack Picture: Adding/Removing an Element

Stack before pushing new number 5512

Stack after pushing new number

Then, stack after popping 4 elements
Stack ADT

- Stack ADT definition:
  - Just the List ADT, but restricted to fewer operations.

  - Consider the list \( A_1, A_2, A_3, \ldots, A_N \).
  - Allow operations:
    - \textit{push} (adds one to stack)
    - \textit{pop} (removes one from stack; usually returns the value)
    - \textit{top} (or \textit{peek}) (returns value of top element; does not pop)
    - \textit{makeEmpty}
    - \textit{isEmpty}

  - There may be other operations, but these are the most common.
Stack Implementation

• Just a List with same methods named differently (and implemented differently)
  • insert is push
  • delete is pop

• So can use linked list. Or can use array.

• When well designed, the user never knows which is behind the scenes.
public class Stack
{
    private ListNode topOfStack;

    public Stack()
    {
        topOfStack = null;
    }

    public void push(int n)
    {
        topOfStack = new ListNode(n, topOfStack);
    }

    // and now could add pop, top, and other methods
}
Alternative Array Implementation

- Use array instead of linked list.
  - Assume array has some fixed size.
  - This fixes the number of objects allowed on the stack!
- Keep track of position on the array.
  - Start with to methodologies.
- To push x onto stack:
  - topOfStack++;
  - set array[topOfStack] = x;
- To pop
  - Read array[topOfStack];
  - topOfStack--;
- To read top
  - return array[topOfStack];

Be sure to test if the array is empty!
Stacks Are Fast!

- Why? Because have list ADT, but have restricted the operations to be very simple.
  - Look at our pop and push methods for the stack – they use linked lists, and we know they are fast.
  - Homework has you estimate the Big-O for array implementation of pop and push.
Stack Applications

- Balancing parentheses in your code.
  - every { must be followed by a }
  - every ( must be followed by a )
  - every [ must be followed by a ]
  - {([])} is valid; {([])} is not valid.
Balance Application

- Write a program that balances!
  - Make empty stack.
  - Read code file.
  - Each time find left-hand parenthesis, push onto stack.
  - Each time find right-hand parenthesis, pop. If popped value is left-hand **of same type** then ok, otherwise report problem.
  - If pop is on an empty stack, then report problem.
  - If reach end of file and stack is not empty, report a problem.
  - Otherwise, everything works!
Balance Example

- Try pushing and popping the following:
  - public class Example
    {
    public static void main(String[] args)
    {
        public static void main(String[] args)
        {
            if(…)
            {
                …
            }
        }
    }

- Now try again but intentionally mess up one of the parentheses, like “main(String[] args)”.
Method Call Application

- Your code makes a series of calls to methods/functions.
  - How does it keep track of where it was before the call?
  - Uses a stack!
Consider the following code inside of a class:

```java
public void doThis() {
    doThat();
}
public void doThat() {
    doTheOther();
}
public void doTheOther() {
    int i = 1;
}
```

Each time a method is called, put it on the stack. When the call is done, pop it. Then knows what method to return to. Go to the method that’s next on the stack.

Now, what happens if doThat() calls doThis()?
Infinite loop. Get a “stack overflow” error.
Queues ADT

- Queues are lists with two changes.
  - insert is done at one end
  - delete is done at the other end

- Think of a line at the cash register. Pretty much any line at all.

- Called First In First Out (FIFO)
  - or “first come first served”
Queue Operators

- **enqueue**
  - inserts at end of the list (called rear)

- **dequeue**
  - deletes (and returns) the element at the start of the list (called front)

Again, think of lines at the grocery store…
Array Queue

- Just have an array and keep track of:
  1. front position
  2. back position
  3. current size of queue (number of elements in queue)

- Handy to have wrap around arrays…
  - see following example.
  - But can run out of room (fill array)
  - Use linked list if that might be a problem, but most queues are usually pretty small because keep calling dequeue at roughly same rate as enqueue.
Array Queue

initial state

enqueue(1)

front

back

front
Array Queue

1 3
   back
   enqueue(3)

2 4
  back
  dequeue(), which returns 2

1 3
   front

2 4
   front
Array Queue

dequeue(), which returns 4

1 3
2 4

front

dequeue(), which returns 1

1 3
2 4

front
Array Queue

dequeue(), which returns 3

Queue is now empty!
public void enqueue (int n) {
    back++;  // Don’t forget to modulo by the array size!
    theArray[back] = n;
    currentSize++;
}

Queue examples

- print jobs on your computer
- software handling phone calls when all operators are busy

- whole branch of mathematics called “queuing theory” (wow check out those vowels!)
  - finds probability of having certain wait time in line
  - probability of having certain length line
  - frequently solve these problems with simulations using our data structure

*Phenomenal* number of applications!