CSCI 136
Data Structures & Advanced Programming

Lecture 19

Spring 2018

Instructors: Bill & Jon

Administrative Details

- Lab 6: PostScript is today
  - Individual lab this week
  - GitHub repositories are ready
  - Any questions?
    - <Review switch statements>

Last Time

- Iterators
  - General purpose mechanism for traversals
- `Iterator` interface (Java)
- `AbstractIterator` class (structure5)
  - Adds `get()` and `reset()`

Today's Outline

- Nifty Iterators
- Bit operations
  - Useful in general, but required for Lab 7
- Return midterm exams

SkipIterator.java

- Goal:
  - Take an iterator `it` and a value `val`
  - Return sequential values from `it` as long as they don't match `val`
- Implementation:
  - `next()` and `hasNext()`
  - What if last value in `it` is equal to `val`?

ReverseIterator.java

- Goal:
  - Take an iterator `it` and return its values in opposite order
- Implementation:
  - Problem: Iterators progress in one direction only
    - `next()` but no `previous()`
  - Any ideas?

Biterator.java

- Goal:
  - Take a number `n`, and yield its bits (0 or 1) from least significant bit to most significant bit
- Implementation:
  - Think back to Lab 3

Representing Numbers

- Humans usually think of numbers in base 10
- But even though we write `int x = 23;` the computer stores `x` as a sequence of `1`s and `0`s
- Recall Lab 3:

```
public static String numInBinary(int n) {
        if (n <= 1)
            return "" + n%2;

        return printInBinary(n/2)+n%2;
}
```

- 00000000 00000000 00000000 00010111

Bitwise Operations

- We can use *bitwise* operations to manipulate the `1`s and `0`s in the binary representation
  - Bitwise 'and': `&`
  - Bitwise 'or': `|`
- Also useful: bit shifts
  - Bit shift left: `<<`
  - Bit shift right: `>>`

& and |

- Given two integers a and b, the *bitwise or* expression `a | b` returns an integer s.t.
  - At each bit position, the result has a `1` if that bit position had a `1` in EITHER a OR b
  - `3 | 6 = ?`
- Given two integers a and b, the *bitwise and* expression `a & b` returns an integer s.t.
  - At each bit position, the result has a `1` if that bit position had a `1` in BOTH a AND b
  - `3 & 6 = ?`

>> and <<

- Given two integers a and i, the expression
  (a << i) returns (a * 2^i)
  - Why? It shifts all bits left by i positions
  - 1 << 4 = ?
- Given two integers a and i, the expression
  (a >> i) returns (a / 2^i)
  - Why? It shifts all bits right by i positions
  - 1 >> 4 = ?
  - 97 >> 3 = ? (97 = 1100001)
- Be careful about shifting left and "overflow"!!!

Revisiting numInBinary(int n)

- How would we rewrite a recursive `numInBinary` using bit shifts and bitwise operations?

```
public static String numInBinary(int n) {
    if (n <= 1)
        return "" + n;
    return numInBinary(n >> 1) + (n & 1);
}
```

Revisiting numInBinary(int n)

- How would we write an iterative `printInBinary` using bit shifts and bitwise operations?

```
public static String printInBinary(int n,
                                   int width) {
    String result = "";
    for(int i = 0; i < width; i++)
        if ((n & (1<<i)) == 0)
            result = 0 + result;
        else
            result = 1 + result;
    return result;
}
```

BIterator.java

- Goal:
  - Take a number `n`, and yield its bits (0 or 1) from least significant bit to most significant bit
- Implementation:
  - Store `n`
  - Each `next()` isolates the LSB and shifts
  - `hasNext()`?
  - `reset()`?

General Rules for Iterators

1. Understand order of data structure
2. **Always call hasNext() before calling next()!!!**
3. Use remove with caution!
4. Don't add to structure while iterating:
   see `TestIterator.java`