CSCI 136: Data Structures and Advanced Programming
Lecture 21
More iterators

Instructor: Dan Barowy
Williams

Topics

• Iterators
• Integer representation

Your to-dos

1. Read before Fri: Bailey, Ch 12.6-12.9.
2. Lab 7 (partner lab), due Tuesday 4/19 by 10pm.

Announcements

  Organized by Prof. Kelly Shaw and CoSSAC.
  Wednesday, April 13 from 7-7:45pm in TBL 211.
  “Extra special snacks” provided by CoSSAC afterward in the Eco Cafe.
• This Friday’s colloquium: CS pre-registration info session.
The bits of an integer

An integer is represented in computer memory as a sequence of bits, each having a value of either 0 or 1. This representation is called binary.

Binary is number system where each digit can take one of two values; i.e., the base of the system is 2.

You are probably more familiar with the base 10 number system, aka decimal.

Any integer can be represented in either system.

Java int

The int data type in Java has 32 bits.

00000000 00000000 00000000 00010111

is the number 23.

\[
\begin{align*}
(00000000000000000000000000010111)_2 &= (0 \times 2^{31}) + (0 \times 2^{30}) + (0 \times 2^{29}) + (0 \times 2^{28}) \\
+ (0 \times 2^{27}) + (0 \times 2^{26}) + (0 \times 2^{25}) + (0 \times 2^{24}) \\
+ (0 \times 2^{23}) + (0 \times 2^{22}) + (0 \times 2^{21}) + (0 \times 2^{20}) \\
+ (0 \times 2^{19}) + (0 \times 2^{18}) + (0 \times 2^{17}) + (0 \times 2^{16}) \\
+ (0 \times 2^{15}) + (0 \times 2^{14}) + (0 \times 2^{13}) + (0 \times 2^{12}) \\
+ (0 \times 2^{11}) + (0 \times 2^{10}) + (0 \times 2^9) + (0 \times 2^8) \\
+ (0 \times 2^7) + (0 \times 2^6) + (0 \times 2^5) + (1 \times 2^4) \\
+ (0 \times 2^3) + (1 \times 2^2) + (1 \times 2^1) + (1 \times 2^0) \\
= (23)_{10}
\end{align*}
\]
Bitwise Operations

We can use bitwise operations to manipulate the 1s and 0s 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 \( b \)

- \( 3 \ | \ 6 = ? \)
  \[
  011 \ | \ 110 = 111
  \]

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 \( b \)

- \( 3 \ & \ 6 = ? \)
  \[
  011 \ & \ 110 = 010
  \]

>> and <<

Given two integers \( a \) and \( i \), the expression \( (a << i) \) returns \( (a \ * \ 2^i) \)
• Why? It shifts all bits left by \( i \) positions

- \( 1 << 4 = ? \)
  \[
  00001 \ << 4 = 10000
  \]

Given two integers \( a \) and \( i \), the expression \( (a >> i) \) returns \( (a \ / \ 2^i) \)
• Why? It shifts all bits right by \( i \) positions

- \( 1 >> 4 = ? \)
  \[
  00001 \ >> 4 = 00000
  \]
- \( 97 >> 3 = ? \)
  \[
  1100001 \ >> 3 = 1100
  \]

Iterators
**Iteration**

Iteration is the repetition of a process in order to generate a (possibly unbounded) sequence of outcomes. Each repetition of the process is a single iteration, and the outcome of each iteration is then the starting point of the next iteration.

Example.

```java
List<Double> ls = new SinglyLinkedList<>();
// _ initialize ls _
double sum = 0.0;
for (double d : ls) {
    sum += d;
}
```

```
100 → 101 → 102 → Ø
```

```
sum
0

d
0
```
Example.

```java
List<Double> ls = new SinglyLinkedList<>();
// _ initialize ls _
double sum = 0.0;
for (double d : ls) {
    sum += d;
}
```

```
head

100 ➔ 101 ➔ 102 Ø

current

Iterable<Double>

sum

100

d

100
```

```
head

100 ➔ 101 ➔ 102 Ø

current

Iterable<Double>

sum

201

d

101
```

```
head

100 ➔ 101 ➔ 102 Ø

current

Iterable<Double>

sum

303

d

102
```

```
head

100 ➔ 101 ➔ 102 Ø

current

Iterable<Double>

sum

303

d

102
```
Example.

```java
List<Double> ls = new SinglyLinkedList<>();
// _initialize ls _
double sum = 0.0;
for (double d : ls) {
    sum += d;
}
```

What’s an `Iterator<T>`?

```
public interface Iterator<E> {
    boolean hasNext();
    E next();
    …
}
```

It’s a stateful object that lets you iterate through a data structure.

A bit iterator

Suppose we want to do the following:

On each iteration, get the next most significant bit, starting initially with the least significant bit.

`BIterator` to the rescue.

Recap & Next Class

**Today:**

- Iterators
- Number representations

**Next class:**

- Tree ADT