Lecture 20

Trees II

- Lab 6 — Preview
  - Bitwise Operations
  - Gray Code
- Exploring Binary Trees
  - Challenge 1
Lab 6 — Preview
8.7 Laboratory: The Two-Towers Problem

Objective. To investigate a difficult problem using Iterators.

Discussion. Suppose that we are given $n$ uniquely sized cubic blocks and that each block has a face area between 1 and $n$. Build two towers by stacking these blocks. How close can we get the heights of the two towers? The following two towers built by stacking 15 blocks, for example, differ in height by only 129 millions of an inch (each unit is one-tenth of an inch):

```
4 5 6 7 9 10 14
3 8 11 12 13 15
```

Still, this stacking is only the second-best solution! To find the best stacking, we could consider all the possible configurations.

We do know one thing: the total height of the two towers is computed by summing the heights of all the blocks:

$$h = \sum_{i=1}^{n} \sqrt{i}$$

If we consider all the subsets of the $n$ blocks, we can think of the subset as the set of blocks that make up, say, the left tower. We need only keep track of that subset that comes closest to $h/2$ without exceeding it.

In this lab, we will represent a set of $n$ distinct objects by a Vector, and we will construct an Iterator that returns each of the $2^n$ subsets.

Computer Science CS136 (Fall 2021)
Duane Bailey & Aaron Williams
Laboratory 6
The Two Towers Problem

Suppose that we are given $n$ uniquely sized cubic blocks and that each block has a face area between 1 and $n$. Build two towers by stacking these blocks. How close can we get the heights of the two towers? The following two towers built by stacking 15 blocks, for example, differ in height by only 129 millionths of an inch (each unit is one-tenth of an inch):

```
4 5 6 7 9 10 14
3 8 11 12 13 15
```

Still, this stacking is only the second-best solution! To find the best stacking, we could consider all the possible configurations.

We do know one thing: the total height of the two towers is computed by summing the heights of all the blocks:

$$h = \sum_{i=1}^{n} \sqrt{i}$$

If we consider all the subsets of the $n$ blocks, we can think of the subset as the set of blocks that make up, say, the left tower. We need only keep track of that subset that comes closest to $h/2$ without exceeding it.

In this lab, we will represent a set of $n$ distinct objects by a Vector, and we will construct an Iterator that returns each of the $2^n$ subsets.

This lab (right) is a version of the classic Two Towers Problem from the textbook (left). But the extension is new this semester.
Each possible solution can be represented as a subset or a binary string or as an integer!

- **Integer 121** ⇒ **binary string** 01111001 ⇒ **subset** \{7, 6, 5, 4, 1\}.

---

Also suppose that we have eight blocks, labeled 1, 2, \ldots, 8, and we arbitrarily choose blocks 1, 4, 5, 6, and 7 as our subset:

If we filled in the corresponding boxes in for each chosen block (i.e., if the block \(i\) is chosen, we put a 1 in spot \(i - 1\) since our blocks are 1-indexed), our paper strip looks like this:

Conveniently, we can think of the combination of 0s and 1s on strip of paper as the digits of an n-bit binary number. For example, the binary number 01111001 is the decimal number 121, since 01111001 is \(2^6 + 2^5 + 2^4 + 2^3 + 2^0\).

(This link provides some helpful illustrations, but is not the lab handout for this semester.)
Bitwise Operations
Bitwise Operations: Isolating a Bit

How can you test if a particular bit is set to 0 or 1 within the binary representation of an integer?

For example, in the integer 121 is the 5th bit 1 or 0?

- An int in Java has 32 bits (4 bytes) and we index them as \( b_{31} \ldots b_1 b_0 \) or just \( b_7 \ldots b_1 b_0 \) in a byte.

This is also known as isolating the bit.

Each bit is worth a power of 2. More specifically, the \( i \)th bit has value \( 2^i \). For example,

\[
121 = 0 \cdot 2^7 + 1 \cdot 2^6 + 1 \cdot 2^5 + 1 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0
\]

\[
= 2^6 + 2^5 + 2^4 + 2^3 + 2^0
\]

\[
= 64 + 32 + 16 + 8 + 1
\]

In particular, the 5th bit is worth \( 2^5 = 32 \). In other words, \( 32_{10} = 00100000_2 \). So we isolate it by AND (\&) with 32.

To obtain 32 we could use repeated multiplication by 2.

Or we can use bit-shifting which move bits to the left or right.

For example, \( 1 \ll 5 \) gives 32 in Java, where 1 is the value that is shifted and 5 is the number of positions to shift its bits.

\[
121 \& (1 \ll 5) = 121 \& 32 = 32 \neq 0
\]

so the 5th bit is 1.
Extension
The extension involves using a Gray code to speed up the exhaustive computation.
We’ll take a 15 minute diversion into Gray codes. It is posted as 20-gray.pdf and is not testable material.
Exploring Binary Trees
Challenge: Exploring Binary Trees (Part 1)

How can we count the number of leaves in a binary tree?

Think about this for 1 minute.
Then discuss it with your neighbor for 2 minutes.

This binary tree has 4 leaves.

Hints:
- Consider a recursive algorithm.
- What are the base cases?
Counting the number of leaves.

Note: We can visualize the return values in the nodes.

- What order would these values be filled in during the algorithm?
Challenge: Exploring Binary Trees (Part 2)

How can we determine the following values in a binary tree?

- The total number of nodes.
- The height of the tree.
- The smallest level that has a leaf.
- The number of left links that are used.

Think about this for 1 minute. Then discuss it with your neighbor for 4 minutes.

This binary tree has 8 total nodes.
It has height 3 (counting from 0).
The smallest level of a leaf is 2.
It has 5 left links in total.

What other quantities could we try to count?

-
// TODO
height(root)
    // Base case:
    if then
        return
Challenge: Exploring Binary Trees (Part 3)

How could we print out a nice text representation of a binary tree?

Think about this for 1 minute.

This binary tree could be printed out as

Questions:
- What do you interpret *nice* to mean?
- What values would you want to compute?