Bitwise Operators & Huffman Coding

Assignment 12

Binary and Hexadecimal Review

An integer is composed of 4 bytes

Byte 3	Byte 2	Byte 1	Byte 0
MSB			LSB

Binary and Hexadecimal Review

A byte is composed of 8 bits (or 2 4-bit "nibbles")

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0

MSB LSB

e.g. 0010 0110

Nibble 1	Nibble 0

e.g.: 0x26

Binary and Hexadecimal Review

- Binary data is often shown in hexadecimal
- Each digit is a 4-bit nibble

0x12 = 18 decimal (1 byte)

0x48ee = 18,670 (2 bytes)

0x1FF64890 = a big number (4 bytes)

Binary and Hexadecimal Review

- Each digit is 4 bits
- Values in range [0-15] or [0..9, A-F]

0110 1110 = 0x6E

binary	hex	binary	hex
0000	0	1000	8
0001	1	1001	9
0010	2	1010	A
0011	3	1011	B
0100	4	1100	C
0101	5	1101	D
0110	6	1110	E
0111	7	1111	F

Bitwise Operations

- Shifting
  - >>, <<, and <<<
  - Moves bits left or right
- Masking
  - The bitwise “and” operation: &
  - Used to set bits to zero or check if bits are set
- Setting
  - The bitwise “or” operation: |
  - Used to set bits to 1

Bitwise Shifting (<<, >>, >>>)

- Moves bits left or right by a specific number of bits
- << shifts bits to the left
- >> shifts bits to the right with sign extension
- >>> shifts to the right *without* sign extension

Shifting

```
byte n = 73; // n=0100 1001
n = n >> 2;  // n=0001 0010
n = n << 3;  // n=1001 0000
n = n >> 2;  // n=1110 0100
n = n >>> 2; // n=0011 1001
```

Mask (&)

```
byte n=0xe3;  // n=227
n = n & 0xf0; //   1110 0011
              // & 1111 0000
              // ___________
              //   1110 0000
```

Set (|)

```
byte n=0x11;  // n=17
n = n | 0xc5; //   0001 0001
              // | 1100 0101
              // ___________
              //   1101 0101
```

Checking bit “n”

```
int checkBit(int b, int n)
{
    return (b >> n) & 1;
}
```

Check bit example:checkBit(0xEA,5)

```
n = 1110 1010
n >>> 5 = 0000 0111
n & 1 = 0000 0001


checkBit(0xe3, 5) == 1
The bit is set!
```

Check bit example:checkBit(0xEA,5)

```
n = 1110 1010
n >>> 5 = 0000 0111
n & 1 = 0000 0001


checkBit(0xe3, 5) == 1
The bit is set!
```

Binary File IO

- File must be read as binary data
- FileInputStream
  - Similar to FileReader
  - `FileInputStream(String fileName)`
- Reads one byte at a time
  - `read()` - returns the next byte of data from the stream
  - Returns -1 when end of file reached

FileInputStream - a 3-byte file

byte 0	byte 1	byte 2
0101 0011	1110 1001	0001 1101

```
.read() → 0x53
.read() → 0xE9
.read() → 0x1D
.read() → -1
```

Binary File IO - read bit

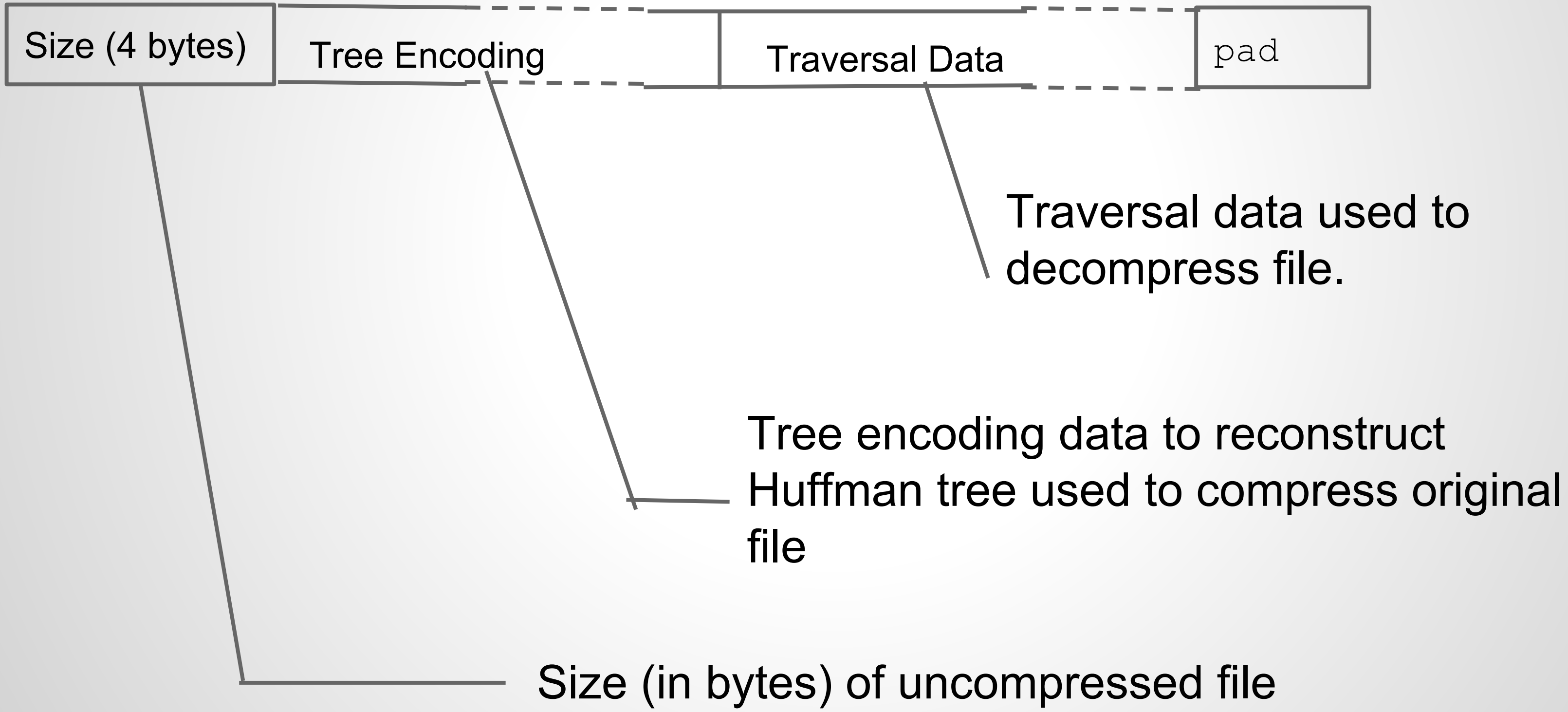
- Need method to read one bit at a time
- Java only provides a method to read bytes
- Read 1 byte
- Keep track of how many bits have been read
- Move to next bit after each call to readBit()
- Once all 8 bits have been read, read new byte

Binary File IO - read byte

- Bytes may not be aligned on 8-bit boundary
- To read a byte, read 8 bits individually
- Combine them with a shift and bitwise “or”

```
byte b = 0;
for (int i=0; i<8; i++) {
   b = b << 1;
   b = b | readBit();
}
```

Compressed File Format



short.txt.huff: original file size

$\boxed{\texttt{0000 0005}}$ 5094 3a0e 98

Size

```
int cnt = 0;
cnt = cnt | fs.read() << 24;
cnt = cnt | fs.read() << 16;
cnt = cnt | fs.read() << 8;
cnt = cnt | fs.read() << 0;
```

short.txt.huff: Building Tree

```
0000 0005 5094 3a0e 98
0101 0000 1001 0100 0011 1010 0000...
```

Internal node

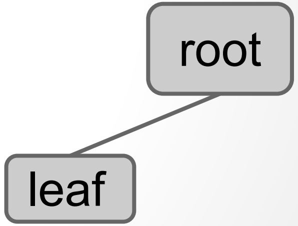
root

short.txt.huff: Building Tree

0000 0005 5094 3a0e 98

0101 0000 1001 0100 0011 1010 0000…

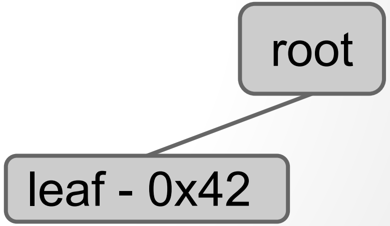
leaf node



short.txt.huff: Building Tree

```
0000 0005 5094 3a0e 98
0101 0000 1001 0100 0011 1010 0000...
```

symbol (0x42)

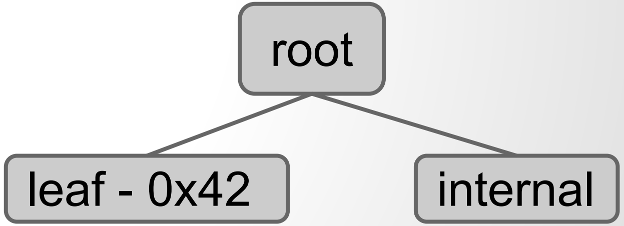


short.txt.huff: Building Tree

0000 0005 5094 3a0e 98

0101 0000 1001 0100 0011 1010 0000...

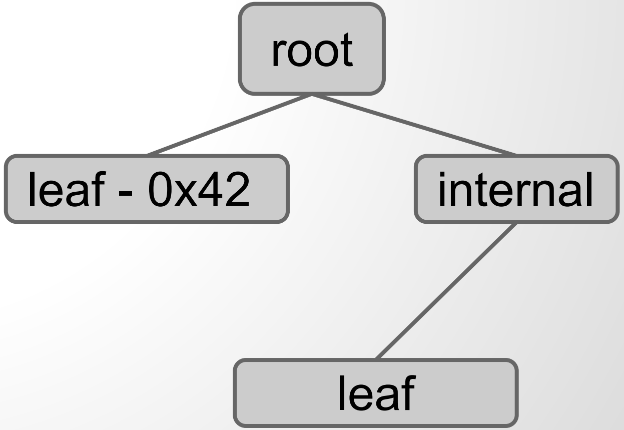
internal node



short.txt.huff: Building Tree

```
0000 0005 5094 3a0e 98
0101 0000 1001 0100 0011 1010 0000...
```

leaf node

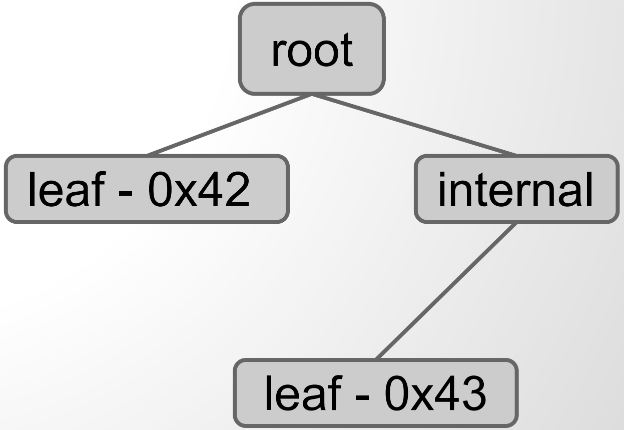


short.txt.huff: Building Tree

```
0000 0005 5094 3a0e 98
```

```
0101 0000 1001 0100 0011 1010 0000
1110...
```

symbol
data

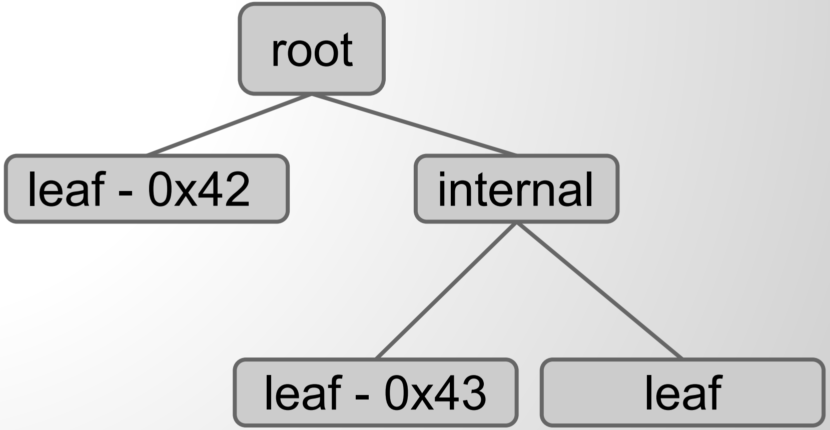


short.txt.huff: Building Tree

leaf

0000 0005 5094 3a0e 98

0101 0000 1001 0100 0011 1010 0000 1110 ...



short.txt.huff: Building Tree

symbol

`0000 0005 5094 3a0e 98`

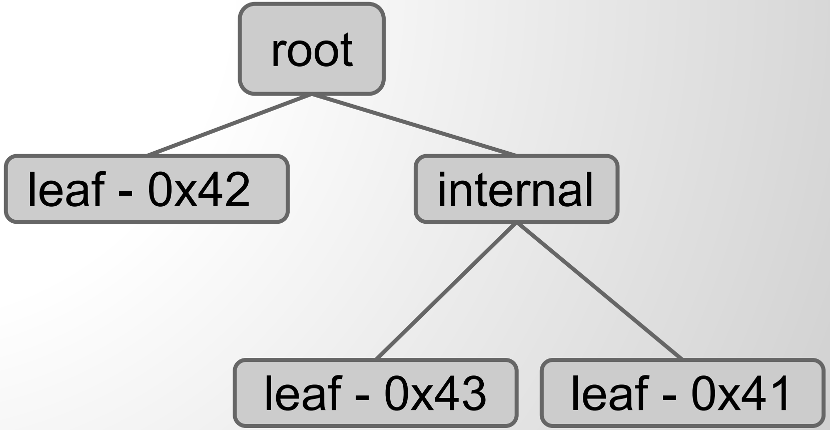
`0101 0000 1001 0100 0011 1010 0000 1110 ...`

ASCII

0x41 = ‘A’

0x42 = ‘B’

0x43 = ‘C’



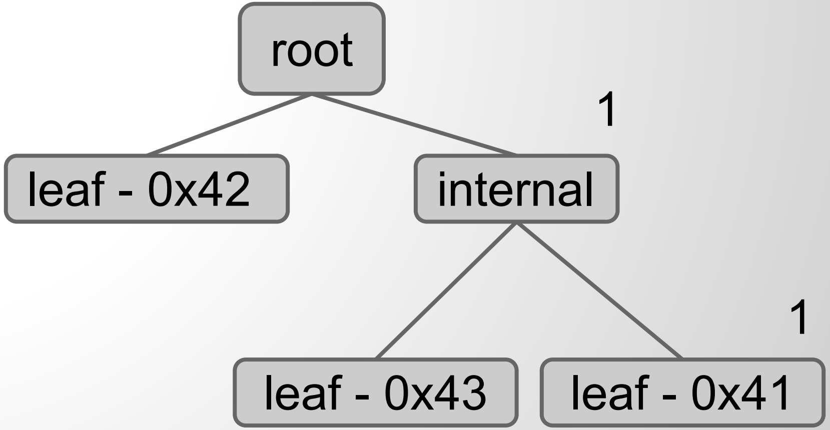
short.txt.huff: Decompress

0000 0005 5094 3a0e 98

…0100 0011 1010 0000 1$\boxed{11}$0 1001 1000

count=1

“A”



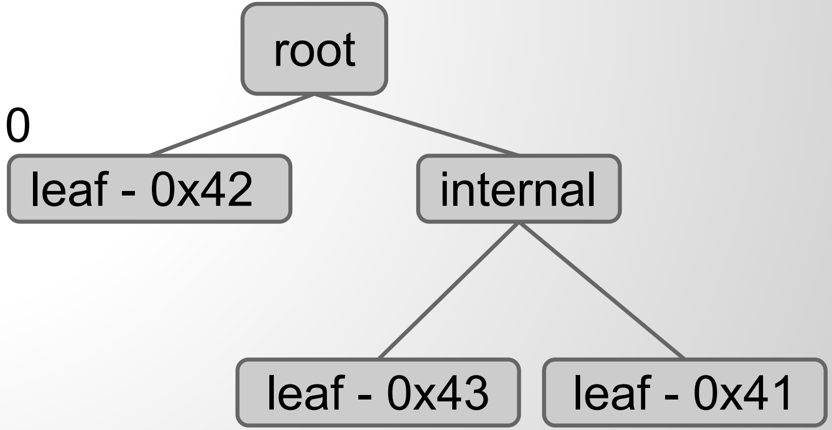
short.txt.huff: Decompress

0000 0005 5094 3a0e 98

…0100 0011 1010 0000 1110 1001 1000

count=2

"AB"



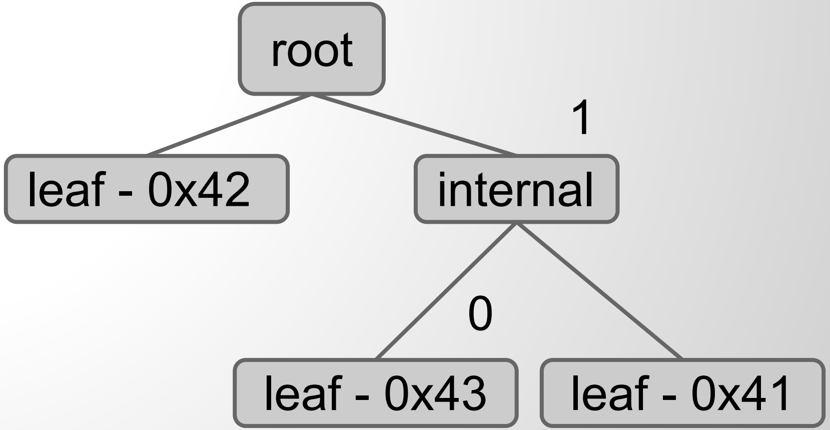
short.txt.huff: Decompress

0000 0005 5094 3a0e 98

…0100 0011 1010 0000 1110 $\boxed{10}$01 1000

count=3

“ABC”



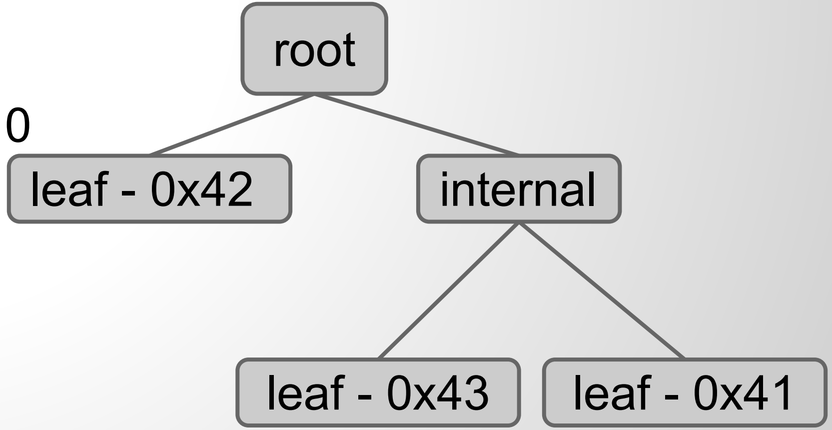
short.txt.huff: Decompress

0000 0005 5094 3a0e 98

…0100 0011 1010 0000 1110 1001 1000

count=4

“ABCB”



short.txt.huff: Decompress

0000 0005 5094 3a0e 98

…0100 0011 1010 0000 1110 1001 1000

count=5

“ABCBA”