

Data representation: Bits, Bytes, Integers

Yipeng Huang

Rutgers University

October 6, 2026

Data representation: Bits, Bytes, Integers

Announcements

Canvas timed quiz 4 and programming assignment 2

Bits and bytes

Why binary

Decimal, binary, octal, and hexadecimal

Representing characters

Bitwise operations

Integers and basic arithmetic

Representing negative and signed integers

Fractions and fixed point representation

Programming assignment 2: Graphs, trees, queues, hashes

Using `graphutils.h`

`bstLevelOrder.c`: Level order traversal of a binary search tree

Binary search tree: `BSTNode`, `insert()`, `delete()`

Linked list implementation of a queue: `QueueNode`, `Queue`, `enqueue()`, `dequeue()`

`matChainMul.c`: Minimum number of multiplies needed for matrix chain multiplication

Data representation: Bits, Bytes, Integers

Announcements

Canvas timed quiz 4 and programming assignment 2

Bits and bytes

Why binary

Decimal, binary, octal, and hexadecimal

Representing characters

Bitwise operations

Integers and basic arithmetic

Representing negative and signed integers

Fractions and fixed point representation

Programming assignment 2: Graphs, trees, queues, hashes

Using `graphutils.h`

`bstLevelOrder.c`: Level order traversal of a binary search tree

Binary search tree: `BSTNode`, `insert()`, `delete()`

Linked list implementation of a queue: `QueueNode`, `Queue`, `enqueue()`, `dequeue()`

`matChainMul.c`: Minimum number of multiplies needed for matrix chain multiplication

Canvas timed quiz 4 and programming assignment 2

Quiz 4

1. Spanning Friday 10/2 - Friday 10/16.
2. 60 minutes.
3. Three tries.
4. Memory bugs and bits, bytes, and integers.

Programming assignment 2

1. Due Tuesday 10/13.
2. Data structures mastery.

Memory Debugging Tools: ASAN vs. Valgrind

▶ **Address Sanitizer (ASAN):**

- ▶ A compile-time tool used during development.
- ▶ Detects memory errors at runtime and provides exact source line traces.
- ▶ Enabled via compiler flags (included in course `Makefile`).

▶ **Valgrind:**

- ▶ A virtual machine that simulates program execution to trace memory calls.
- ▶ Used as a final sanity check on production-ready binaries.
- ▶ **Note:** Cannot be run simultaneously on a binary compiled with ASAN.

Data representation: Bits, Bytes, Integers

Announcements

Canvas timed quiz 4 and programming assignment 2

Bits and bytes

Why binary

Decimal, binary, octal, and hexadecimal

Representing characters

Bitwise operations

Integers and basic arithmetic

Representing negative and signed integers

Fractions and fixed point representation

Programming assignment 2: Graphs, trees, queues, hashes

Using `graphutils.h`

`bstLevelOrder.c`: Level order traversal of a binary search tree

Binary search tree: `BSTNode`, `insert()`, `delete()`

Linked list implementation of a queue: `QueueNode`, `Queue`, `enqueue()`, `dequeue()`

`matChainMul.c`: Minimum number of multiplies needed for matrix chain multiplication

Data representation: Bits, Bytes, Integers

Announcements

Canvas timed quiz 4 and programming assignment 2

Bits and bytes

Why binary

Decimal, binary, octal, and hexadecimal

Representing characters

Bitwise operations

Integers and basic arithmetic

Representing negative and signed integers

Fractions and fixed point representation

Programming assignment 2: Graphs, trees, queues, hashes

Using `graphutils.h`

`bstLevelOrder.c`: Level order traversal of a binary search tree

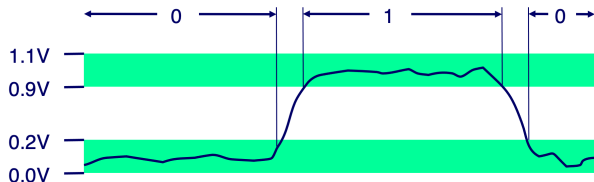
Binary search tree: `BSTNode`, `insert()`, `delete()`

Linked list implementation of a queue: `QueueNode`, `Queue`, `enqueue()`, `dequeue()`

`matChainMul.c`: Minimum number of multiplies needed for matrix chain multiplication

Everything is bits

- Each bit is 0 or 1
- By encoding/interpreting sets of bits in various ways
 - Computers determine what to do (instructions)
 - ... and represent and manipulate numbers, sets, strings, etc...
- Why bits? Electronic Implementation
 - Easy to store with bistable elements
 - Reliably transmitted on noisy and inaccurate wires



Data representation: Bits, Bytes, Integers

Announcements

Canvas timed quiz 4 and programming assignment 2

Bits and bytes

Why binary

Decimal, binary, octal, and hexadecimal

Representing characters

Bitwise operations

Integers and basic arithmetic

Representing negative and signed integers

Fractions and fixed point representation

Programming assignment 2: Graphs, trees, queues, hashes

Using `graphutils.h`

`bstLevelOrder.c`: Level order traversal of a binary search tree

Binary search tree: `BSTNode`, `insert()`, `delete()`

Linked list implementation of a queue: `QueueNode`, `Queue`, `enqueue()`, `dequeue()`

`matChainMul.c`: Minimum number of multiplies needed for matrix chain multiplication

Decimal, binary, octal, and hexadecimal

Decimal	Binary	Octal	Hexadecimal	Decimal	Binary	Octal	Hexadecimal
0	0b0000	000	0x0	8	0b1000	0010	0x8
1	0b0001	001	0x1	9	0b1001	0011	0x9
2	0b0010	002	0x2	10	0b1010	0012	0xA
3	0b0011	003	0x3	11	0b1011	0013	0xB
4	0b0100	004	0x4	12	0b1100	0014	0xC
5	0b0101	005	0x5	13	0b1101	0015	0xD
6	0b0110	006	0x6	14	0b1110	0016	0xE
7	0b0111	007	0x7	15	0b1111	0017	0xF

In C, format specifiers for printf() and fscanf():

1. decimal: '%d'
2. binary: none
3. octal: '%o'
4. hexadecimal: '%x'

Decimal, binary, octal, and hexadecimal

How to represent the range of unsigned char in each?

Unsigned char is one byte, 8 bits.

1. decimal: 0 to 255
2. binary: 0b0 to 0b11111111
3. octal: 0 to 0o377 (group by 3 bits)
4. hexadecimal: 0x00 to 0xFF (group by 4 bits)

Often encountered use of hexadecimal: RGB colors

Red, green, blue values ranging from 0-255

			
#000000	#FFFFFF	#6A757C	???
			#CC0033

Often encountered use of hexadecimal: RGB colors

Red, green, blue values ranging from 0-255

#000000	#FFFFFF	#6A757C	#CC0033

Data representation: Bits, Bytes, Integers

Announcements

Canvas timed quiz 4 and programming assignment 2

Bits and bytes

Why binary

Decimal, binary, octal, and hexadecimal

Representing characters

Bitwise operations

Integers and basic arithmetic

Representing negative and signed integers

Fractions and fixed point representation

Programming assignment 2: Graphs, trees, queues, hashes

Using `graphutils.h`

`bstLevelOrder.c`: Level order traversal of a binary search tree

Binary search tree: `BSTNode`, `insert()`, `delete()`

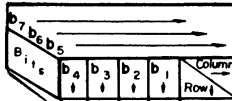
Linked list implementation of a queue: `QueueNode`, `Queue`, `enqueue()`, `dequeue()`

`matChainMul.c`: Minimum number of multiplies needed for matrix chain multiplication

Representing characters

- ▶ char is a 1-byte, 8-bit data type.
- ▶ ASCII is a 7-bit encoding standard.
- ▶ "man ascii" to see Linux manual.
- ▶ Compile and run `ascii.c` to see it in action.
- ▶ Some interesting characters: 7 (bell), 10 (new line), 27 (escape).

USASCII code chart



					0	0	0	0	1	0	1	0	1	1
					0	1	2	3	4	5	6	7		
0	0	0	0	0	NUL	DLE	SP	0	@	P	\	p		
0	0	0	1	1	SOH	DC1	!	1	A	Q	a	q		
0	0	1	0	0	2	STX	DC2	"	2	B	R	b	r	
0	0	1	1	1	3	ETX	DC3	#	3	C	S	c	s	
0	1	0	0	0	4	EOT	DC4	\$	4	D	T	d	t	
0	1	0	1	1	5	ENQ	NAK	%	5	E	U	e	u	
0	1	1	0	0	6	ACK	SYN	&	6	F	V	f	v	
0	1	1	1	1	7	BEL	ETB	'	7	G	W	g	w	
1	0	0	0	0	8	BS	CAN	(	8	H	X	h	x	
1	0	0	1	1	9	HT	EM	)	9	I	Y	i	y	
1	0	1	0	0	10	LF	SUB	*	:	J	Z	j	z	
1	0	1	1	1	11	VT	ESC	+	;	K	[	k	{	
1	1	0	0	0	12	FF	FS	,	<	L	\	l		
1	1	0	1	1	13	CR	GS	-	=	M	]	m	}	
1	1	1	0	0	14	SO	RS	.	>	N	^	n	~	
1	1	1	1	1	15	SI	US	/	?	O	_	o	DEL	

Figure: ASCII character set. Image credit Wikimedia

Data representation: Bits, Bytes, Integers

Announcements

Canvas timed quiz 4 and programming assignment 2

Bits and bytes

Why binary

Decimal, binary, octal, and hexadecimal

Representing characters

Bitwise operations

Integers and basic arithmetic

Representing negative and signed integers

Fractions and fixed point representation

Programming assignment 2: Graphs, trees, queues, hashes

Using `graphutils.h`

`bstLevelOrder.c`: Level order traversal of a binary search tree

Binary search tree: `BSTNode`, `insert()`, `delete()`

Linked list implementation of a queue: `QueueNode`, `Queue`, `enqueue()`, `dequeue()`

`matChainMul.c`: Minimum number of multiplies needed for matrix chain multiplication

Bitwise operations

Why are bitwise operations important?

- ▶ Network and UNIX settings using bit masks (e.g., umask)
- ▶ Hardware and microcontroller programming (e.g., Arduinos)
- ▶ Instruction set architecture encodings (e.g., ARM, x86)

Bitwise operations

~: bitwise NOT

unsigned char a = 128

$a = 0b1000_0000$

$\sim a = \sim 0b1000_0000$

$= 0b0111_1111$

$= 127$

b	$\sim b$
0	1
1	0

Bitwise operations

&: bitwise AND

$$\begin{aligned} 3 \& 1 &= 0b11 \& 0b01 \\ &= 0b01 \\ &= 1 \end{aligned}$$

a	b	a & b
0	0	0
0	1	0
1	0	0
1	1	1

Bitwise operations

`|`: bitwise OR

$$\begin{aligned} 3|1 &= \text{ob}11|\text{ob}01 \\ &= \text{ob}11 \\ &= 3 \end{aligned}$$

$$\begin{aligned} 2|1 &= \text{ob}10|\text{ob}01 \\ &= \text{ob}11 \\ &= 3 \end{aligned}$$

a	b	a b
0	0	0
0	1	1
1	0	1
1	1	1

Bitwise operations

$\wedge$: bitwise XOR

$$\begin{aligned} 3 \wedge 1 &= \text{ob}11 \wedge \text{ob}01 \\ &= \text{ob}10 \\ &= 2 \end{aligned}$$

a	b	$a \wedge b$
0	0	0
0	1	1
1	0	1
1	1	0

`inplaceSwap.c`: Swapping variables without temp variables.

How does it work?

Don't confuse bitwise operators with logical operators

Bitwise operators

- ▶ ~
- ▶ &
- ▶ |
- ▶ ^

Logical operators

- ▶ !
- ▶ &&
- ▶ ||
- ▶ != (for bool type)

Data representation: Bits, Bytes, Integers

Announcements

Canvas timed quiz 4 and programming assignment 2

Bits and bytes

Why binary

Decimal, binary, octal, and hexadecimal

Representing characters

Bitwise operations

Integers and basic arithmetic

Representing negative and signed integers

Fractions and fixed point representation

Programming assignment 2: Graphs, trees, queues, hashes

Using `graphutils.h`

`bstLevelOrder.c`: Level order traversal of a binary search tree

Binary search tree: `BSTNode`, `insert()`, `delete()`

Linked list implementation of a queue: `QueueNode`, `Queue`, `enqueue()`, `dequeue()`

`matChainMul.c`: Minimum number of multiplies needed for matrix chain multiplication

Data representation: Bits, Bytes, Integers

Announcements

Canvas timed quiz 4 and programming assignment 2

Bits and bytes

Why binary

Decimal, binary, octal, and hexadecimal

Representing characters

Bitwise operations

Integers and basic arithmetic

Representing negative and signed integers

Fractions and fixed point representation

Programming assignment 2: Graphs, trees, queues, hashes

Using `graphutils.h`

`bstLevelOrder.c`: Level order traversal of a binary search tree

Binary search tree: `BSTNode`, `insert()`, `delete()`

Linked list implementation of a queue: `QueueNode`, `Queue`, `enqueue()`, `dequeue()`

`matChainMul.c`: Minimum number of multiplies needed for matrix chain multiplication

Representing negative and signed integers

Ways to represent negative numbers

1. Sign magnitude
2. 1s' complement
3. 2's complement

Representing negative and signed integers

Sign magnitude

Flip leading bit.

Representing negative and signed integers

1s' complement

- ▶ Flip all bits
- ▶ Addition in 1s' complement is sound
- ▶ In this encoding there are 2 encodings for 0
- ▶ -0: 0b1111
- ▶ +0: 0b0000

Representing negative and signed integers

2's complement

signed char	weight in decimal
00000001	1
00000010	2
00000100	4
00001000	8
00010000	16
00100000	32
01000000	64
10000000	-128

Table: Weight of each bit in a signed char type

- ▶ what is the most positive value you can represent? 127
- ▶ what is the most negative value you can represent? -128
- ▶ how to represent -1? 11111111
- ▶ how to represent -2? 11111110

Representing negative and signed integers

2's complement

signed char	weight in decimal
00000001	1
00000010	2
00000100	4
00001000	8
00010000	16
00100000	32
01000000	64
10000000	-128

Table: Weight of each bit in a signed char type

- ▶ MSB: 1 for negative
- ▶ To make a number negative: flip all bits and add 1.
- ▶ Addition in 2's complement is sound

Importance of paying attention to limits of encoding

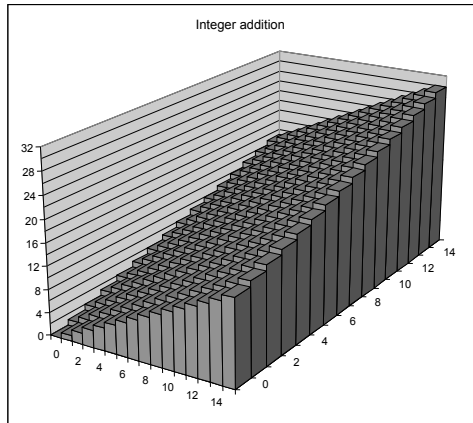


Figure: Image credit: CS:APP

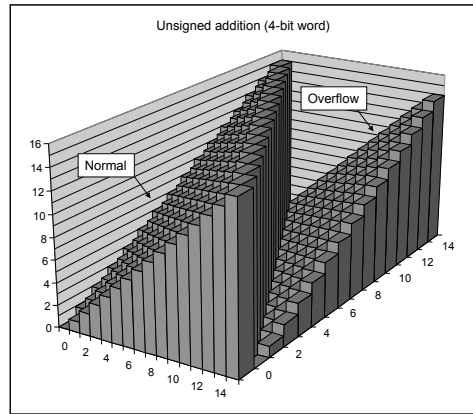


Figure: Image credit: CS:APP

Importance of paying attention to limits of encoding

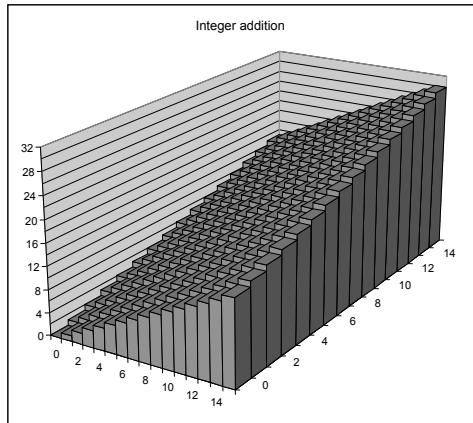


Figure: Image credit: CS:APP

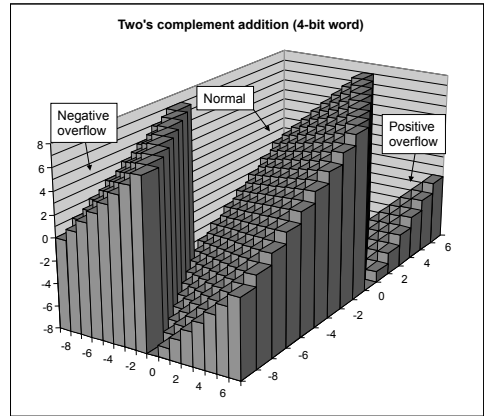


Figure: Image credit: CS:APP

<https://www.theatlantic.com/technology/archive/2014/12/how-gangnam-style-broke-youtube/383389/>

Data representation: Bits, Bytes, Integers

Announcements

Canvas timed quiz 4 and programming assignment 2

Bits and bytes

Why binary

Decimal, binary, octal, and hexadecimal

Representing characters

Bitwise operations

Integers and basic arithmetic

Representing negative and signed integers

Fractions and fixed point representation

Programming assignment 2: Graphs, trees, queues, hashes

Using `graphutils.h`

`bstLevelOrder.c`: Level order traversal of a binary search tree

Binary search tree: `BSTNode`, `insert()`, `delete()`

Linked list implementation of a queue: `QueueNode`, `Queue`, `enqueue()`, `dequeue()`

`matChainMul.c`: Minimum number of multiplies needed for matrix chain multiplication

Unsigned fixed-point binary for fractions

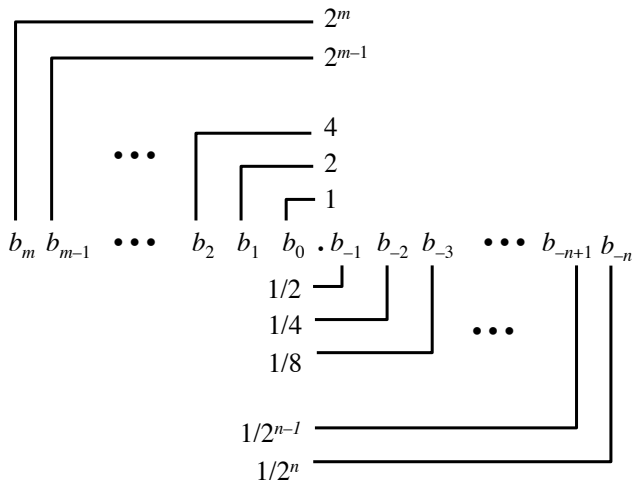


Figure: Fractional binary. Image credit CS:APP

Unsigned fixed-point binary for fractions

unsigned fixed-point char example	weight in decimal
1000.0000	8
0100.0000	4
0010.0000	2
0001.0000	1
0000.1000	0.5
0000.0100	0.25
0000.0010	0.125
0000.0001	0.0625

Table: Weight of each bit in an example fixed-point binary number

- ▶ $.625 = .5 + .125 = 0000.1010_2$
- ▶ $1001.1000_2 = 9 + .5 = 9.5$

Signed fixed-point binary for fractions

signed fixed-point char example	weight in decimal
1000.0000	-8
0100.0000	4
0010.0000	2
0001.0000	1
0000.1000	0.5
0000.0100	0.25
0000.0010	0.125
0000.0001	0.0625

Table: Weight of each bit in an example fixed-point binary number

- ▶ $-.625 = -8 + 4 + 2 + 1 + 0 + .25 + .125 = 1111.0110_2$
- ▶ $1001.1000_2 = -8 + 1 + .5 = -6.5$

Limitations of fixed-point

- ▶ Can only represent numbers of the form $x/2^k$
- ▶ Cannot represent numbers with very large magnitude (great range) or very small magnitude (great precision)

Bit shifting

$\ll$ N Left shift by N bits

- ▶ multiplies by 2^N
- ▶ $2 \ll 3 = 0000_0010_2 \ll 3 = 0001_0000_2 = 16 = 2 * 2^3$
- ▶ $-2 \ll 3 = 1111_1110_2 \ll 3 = 1111_0000_2 = -16 = -2 * 2^3$

$\gg$ N Right shift by N bits

- ▶ divides by 2^N
- ▶ $16 \gg 3 = 0001_0000_2 \gg 3 = 0000_0010_2 = 2 = 16/2^3$
- ▶ $-16 \gg 3 = 1111_0000_2 \gg 3 = 1111_1110_2 = -2 = -16/2^3$

Data representation: Bits, Bytes, Integers

Announcements

Canvas timed quiz 4 and programming assignment 2

Bits and bytes

Why binary

Decimal, binary, octal, and hexadecimal

Representing characters

Bitwise operations

Integers and basic arithmetic

Representing negative and signed integers

Fractions and fixed point representation

Programming assignment 2: Graphs, trees, queues, hashes

Using `graphutils.h`

`bstLevelOrder.c`: Level order traversal of a binary search tree

Binary search tree: `BSTNode`, `insert()`, `delete()`

Linked list implementation of a queue: `QueueNode`, `Queue`, `enqueue()`, `dequeue()`

`matChainMul.c`: Minimum number of multiplies needed for matrix chain multiplication

Programming assignment 2: Graphs, trees, queues, hashes

Programming Assignment 2 parts

1. hashTable: implementing a separate-chaining hash table.
2. edgelist: loading and printing a graph
3. mst: a greedy algorithm
4. solveMaze: needs either DFS (stack) or BFS (queue)
5. findCycle: needs either DFS (stack) or BFS (queue)
6. matChainMul: a dynamic programming approach to multiplying matrices with minimal operations using recursion

Data representation: Bits, Bytes, Integers

Announcements

Canvas timed quiz 4 and programming assignment 2

Bits and bytes

Why binary

Decimal, binary, octal, and hexadecimal

Representing characters

Bitwise operations

Integers and basic arithmetic

Representing negative and signed integers

Fractions and fixed point representation

Programming assignment 2: Graphs, trees, queues, hashes

Using `graphutils.h`

`bstLevelOrder.c`: Level order traversal of a binary search tree

Binary search tree: `BSTNode`, `insert()`, `delete()`

Linked list implementation of a queue: `QueueNode`, `Queue`, `enqueue()`, `dequeue()`

`matChainMul.c`: Minimum number of multiplies needed for matrix chain multiplication

Using `graphutils.h`

- ▶ The adjacency list representation
- ▶ The edgelist representation
- ▶ The query

Data representation: Bits, Bytes, Integers

Announcements

Canvas timed quiz 4 and programming assignment 2

Bits and bytes

Why binary

Decimal, binary, octal, and hexadecimal

Representing characters

Bitwise operations

Integers and basic arithmetic

Representing negative and signed integers

Fractions and fixed point representation

Programming assignment 2: Graphs, trees, queues, hashes

Using `graphutils.h`

`bstLevelOrder.c`: Level order traversal of a binary search tree

Binary search tree: `BSTNode`, `insert()`, `delete()`

Linked list implementation of a queue: `QueueNode`, `Queue`, `enqueue()`, `dequeue()`

`matChainMul.c`: Minimum number of multiplies needed for matrix chain multiplication

Binary search tree

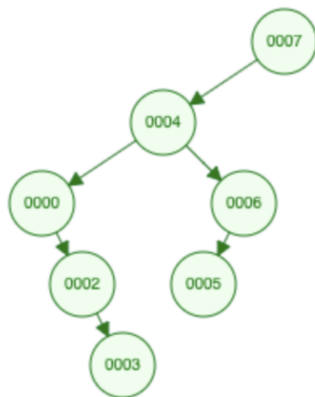


Figure: BST with input sequence 7, 4, 7, 0, 6, 5, 2, 3. Duplicates ignored.

Binary search tree level order traversal

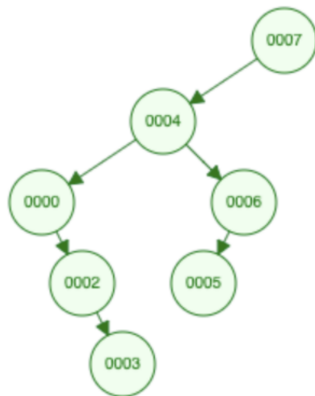


Figure: Level order, left-to-right traversal would return 7, 4, 0, 6, 2, 5, 3.

Binary search tree traversal orders

Breadth-first

- ▶ For example: level-order.
- ▶ Needs a queue (first in first out).
- ▶ Today in class we will build a BST and a Queue.

Depth-first

- ▶ For example: in-order traversal, reverse-order traversal.
- ▶ Needs a stack (first in last out).
- ▶ But in the example code implementation, *where is the stack data structure?*

Data representation: Bits, Bytes, Integers

Announcements

Canvas timed quiz 4 and programming assignment 2

Bits and bytes

Why binary

Decimal, binary, octal, and hexadecimal

Representing characters

Bitwise operations

Integers and basic arithmetic

Representing negative and signed integers

Fractions and fixed point representation

Programming assignment 2: Graphs, trees, queues, hashes

Using `graphutils.h`

`bstLevelOrder.c`: Level order traversal of a binary search tree

Binary search tree: `BSTNode`, `insert()`, `delete()`

Linked list implementation of a queue: `QueueNode`, `Queue`, `enqueue()`, `dequeue()`

`matChainMul.c`: Minimum number of multiplies needed for matrix chain multiplication

typedef

Why types are important

- ▶ Natural language has nouns, verbs, adjectives, adverbs.
- ▶ Type safety.
- ▶ Interpretation vs. compilation.

BSTNode

```
typedef struct BSTNode BSTNode;  
struct BSTNode {  
    int key;  
    BSTNode* l_child; // nodes with smaller key will be in left s  
    BSTNode* r_child; // nodes with larger key will be in right s  
};
```

Data representation: Bits, Bytes, Integers

Announcements

Canvas timed quiz 4 and programming assignment 2

Bits and bytes

Why binary

Decimal, binary, octal, and hexadecimal

Representing characters

Bitwise operations

Integers and basic arithmetic

Representing negative and signed integers

Fractions and fixed point representation

Programming assignment 2: Graphs, trees, queues, hashes

Using `graphutils.h`

`bstLevelOrder.c`: Level order traversal of a binary search tree

Binary search tree: `BSTNode`, `insert()`, `delete()`

Linked list implementation of a queue: `QueueNode`, `Queue`, `enqueue()`, `dequeue()`

`matChainMul.c`: Minimum number of multiplies needed for matrix chain multiplication

QueueNode, Queue

```
// queue needed for level order traversal
typedef struct QueueNode QueueNode;
struct QueueNode {
    BSTNode* data;
    QueueNode* next; // pointer to next node in linked list
};
typedef struct Queue {
    QueueNode* front; // front (head) of the queue
    QueueNode* back; // back (tail) of the queue
} Queue;
```

Let's implement `enqueue()`

<https://visualgo.net/en/queue>

- ▶ First, consider if queue is empty.
- ▶ Then, consider if queue is not empty. Only need to touch back (tail) of the queue.

Let's implement `dequeue()`

`https://visualgo.net/en/queue`

- ▶ First, consider if queue will become empty.
- ▶ Then, consider if queue will not not empty. Only need to touch front (head) of the queue.

Subtle point: why are the function signatures (return, parameters) of `enqueue()` and `dequeue()` the way they are?

Data representation: Bits, Bytes, Integers

Announcements

Canvas timed quiz 4 and programming assignment 2

Bits and bytes

Why binary

Decimal, binary, octal, and hexadecimal

Representing characters

Bitwise operations

Integers and basic arithmetic

Representing negative and signed integers

Fractions and fixed point representation

Programming assignment 2: Graphs, trees, queues, hashes

Using `graphutils.h`

`bstLevelOrder.c`: Level order traversal of a binary search tree

Binary search tree: `BSTNode`, `insert()`, `delete()`

Linked list implementation of a queue: `QueueNode`, `Queue`, `enqueue()`, `dequeue()`

`matChainMul.c`: Minimum number of multiplies needed for matrix chain multiplication

matChainMul.c: Minimum number of multiplies needed for matrix chain multiplication

Learning objectives

- ▶ Review and master recursion.
- ▶ Array subsetting using pointer arithmetic.
- ▶ Using pass-by-reference to return computed results.
- ▶ A new algorithm that most classmates have not seen before.

Cost of multiplying matrices: the number of multiplies

- ▶ $A_{l \times m} \times B_{m \times n}$
- ▶ Needs $l \times m \times n$ number of multiplies
- ▶ (Well-kept secret: fewer multiplications possible, see Strassen's algorithm)

matChainMul.c: Minimum number of multiplies needed for matrix chain multiplication

$$A \times B \times C = \begin{bmatrix} a_{0,0} & a_{0,1} \end{bmatrix}_{1 \times 2} \times \begin{bmatrix} b_{0,0} \\ b_{1,0} \end{bmatrix}_{2 \times 1} \times \begin{bmatrix} c_{0,0} & c_{0,1} \end{bmatrix}_{1 \times 2}$$

Parenthesization 1: 4+4 = 8 multiplies

$$\begin{aligned} A \times (B \times C) &= \begin{bmatrix} a_{0,0} & a_{0,1} \end{bmatrix}_{1 \times 2} \times \begin{bmatrix} b_{0,0}c_{0,0} & b_{0,0}c_{0,1} \\ b_{1,0}c_{0,0} & b_{1,0}c_{0,1} \end{bmatrix}_{2 \times 2} \\ &= \begin{bmatrix} (a_{0,0}b_{0,0}c_{0,0} + a_{0,1}b_{1,0}c_{0,0}) & (a_{0,0}b_{0,0}c_{0,1} + a_{0,1}b_{1,0}c_{0,1}) \end{bmatrix}_{1 \times 2} \end{aligned}$$

Parenthesization 2: 2+2 = 4 multiplies

$$\begin{aligned} (A \times B) \times C &= (a_{0,0}b_{0,0} + a_{0,1}b_{1,0}) \times \begin{bmatrix} c_{0,0} & c_{0,1} \end{bmatrix}_{1 \times 2} \\ &= \begin{bmatrix} (a_{0,0}b_{0,0} + a_{0,1}b_{1,0})c_{0,0} & (a_{0,0}b_{0,0} + a_{0,1}b_{1,0})c_{0,1} \end{bmatrix}_{1 \times 2} \end{aligned}$$

matChainMul.c: Minimum number of multiplies needed for matrix chain multiplication

$$A \times B \times C \times D$$

First partitioning

- ▶ $A(BCD)$; but what is cost of finding (BCD) ? Needs decomposition.
- ▶ $(AB)(CD)$
- ▶ $(ABC)D$; but what is cost of finding (ABC) ? Needs decomposition.

Second partitioning

- ▶ $A(B(CD))$
- ▶ $A((BC)D)$
- ▶ $(AB)(CD)$
- ▶ $(A(BC))D$
- ▶ $((AB)C)D$