Representing and Manipulating Information: Integer operations and fixed point

Yipeng Huang

Rutgers University

February 23, 2023

Table of contents

Programming assignment 2: Queues, trees, and graphs

Programming Assignment 2 parts

1. bstLevelOrder: needs a queue (available in pa2/queue, will discuss today)
2. edgelist: will discuss today
3. isTree: needs DFS (stack)
4. solveMaze: needs BFS (queue)
5. mst: a greedy algorithm
6. findCycle: needs either DFS (stack) or BFS (queue)
7. matChainMul: another dynamic programming problem and prelude to integer operations

Programming assignment 2 & reading assignment

Programming assignment 2

1. Due Friday 2/24.
2. More data structures: queues, BSTs, graphs; solidify managing memory.

Reading assignment: CS:APP Chapters 2.4

1. Preparation for next week
2. All about floating point numbers: A case studying in the design of an engineering standard.

Table of contents

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

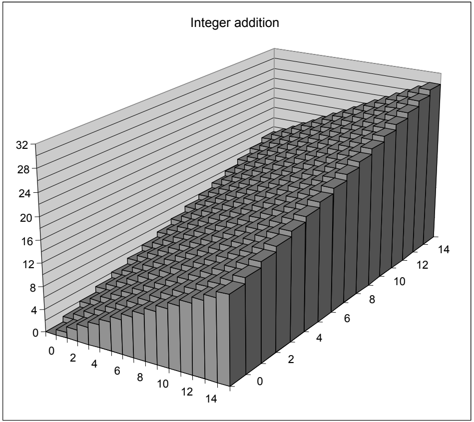

Figure: Image credit: CS:APP

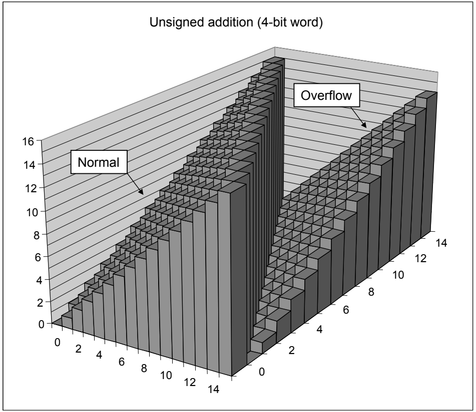

Figure: Image credit: CS:APP

Importance of paying attention to limits of encoding

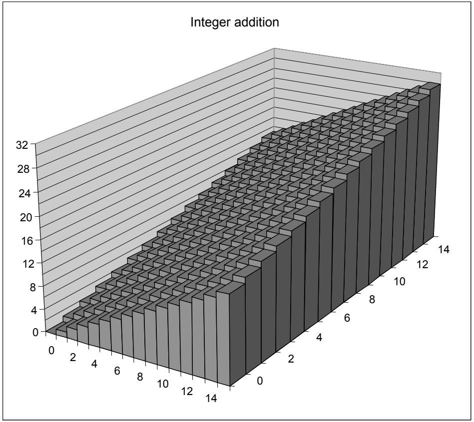

Figure: Image credit: CS:APP

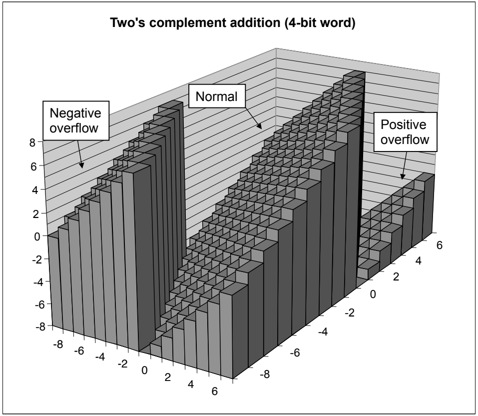

Figure: Image credit: CS:APP

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

`toBin.c`: Printing the binary representation

- Shifting and masking
- Try modifying to print octal.

Bit shifting

$<< N$ Left shift by N bits

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

$>> N$ Right shift by N bits

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

Table of contents

Unsigned fixed-point binary for fractions

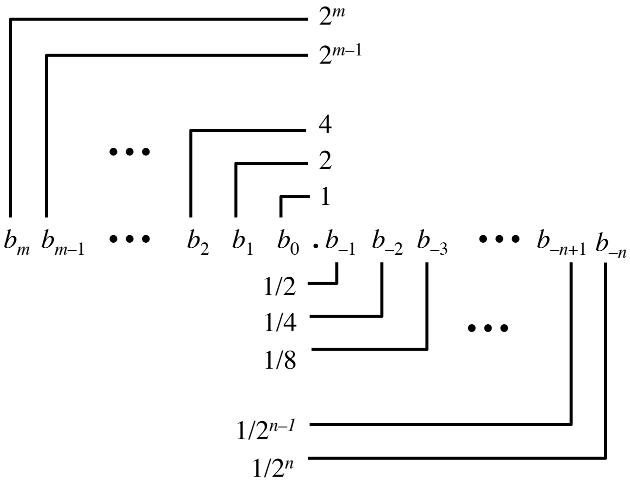

Figure: Fractional binary. Image credit CS:APP

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)

Table of contents

`matChainMul.c`: Minimum number of multiples 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 multiples 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

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

Parenthesization 2: 2+2 = 4 multiplies

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

`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

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