# Representing and Manipulating Information: Two's complement signed integers, fixed point

Yipeng Huang

Rutgers University

February 20, 2024

# Table of contents

# Canvas timed quiz 4 and programming assignment 2

## Programming assignment 2

1. Due Friday 2/23.
2. Graph algorithms and hash table.

# Table of contents

# Representing negative and signed integers

| Ways to represent negative numbers | Pros | Cons |
|---|---|---|
| 1. Sign magnitude | intuitive | Dup 0's, No simple shortcut |
| 2. 1s' complement | intuitive, addition works | Wrap around carry, Dup 0's |
| 3. 2's complement | | |

# 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

# 2's complement.

| most neg number | | zero | largest pos. |
|---|---|---|---|
| 1000_0000 | 1000_0001 | 0000_0000 | 0111_1111 |
| -128 | -127 | | +127 |

| neg. one | pos one |
|---|---|
| 1111_1111 | 0000_0001 |

$-128 + 127 = -1$

```
  1000_0000
+ 0111_1111
-----------
  1111_1111   -1
```

# 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

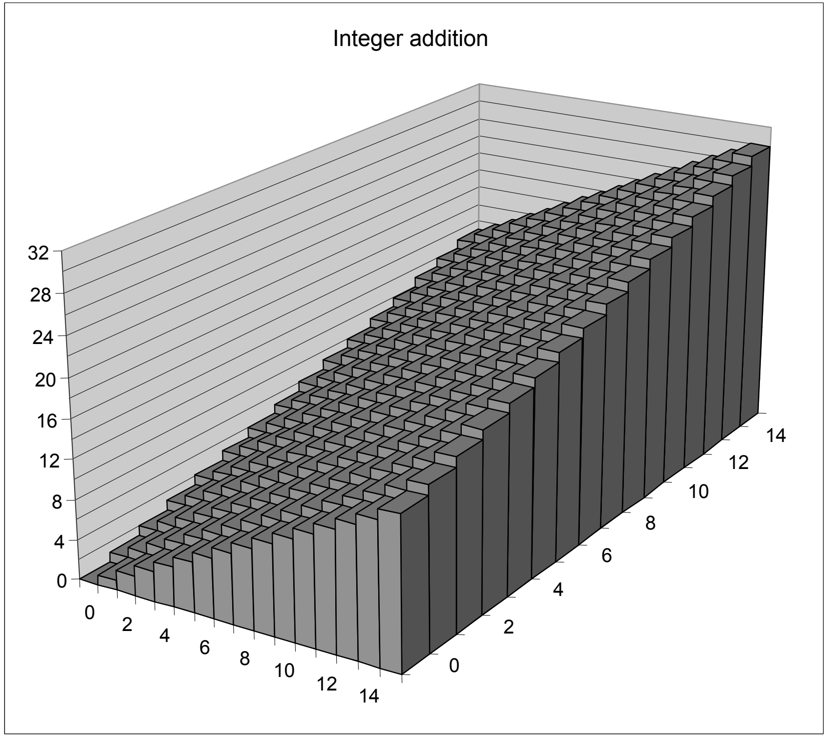


Figure: Image credit: CS:APP

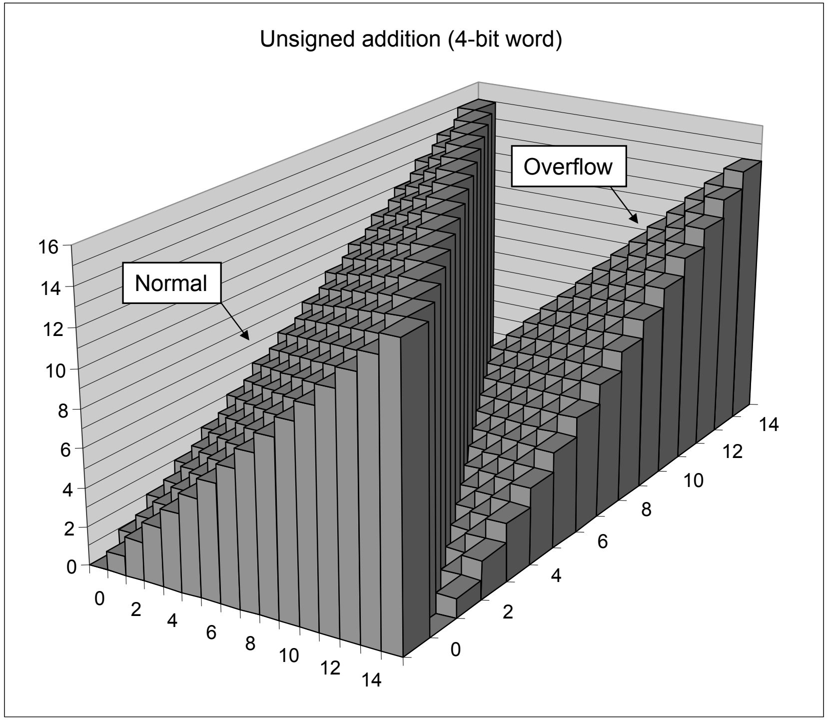


Figure: Image credit: CS:APP

# Importance of paying attention to limits of encoding

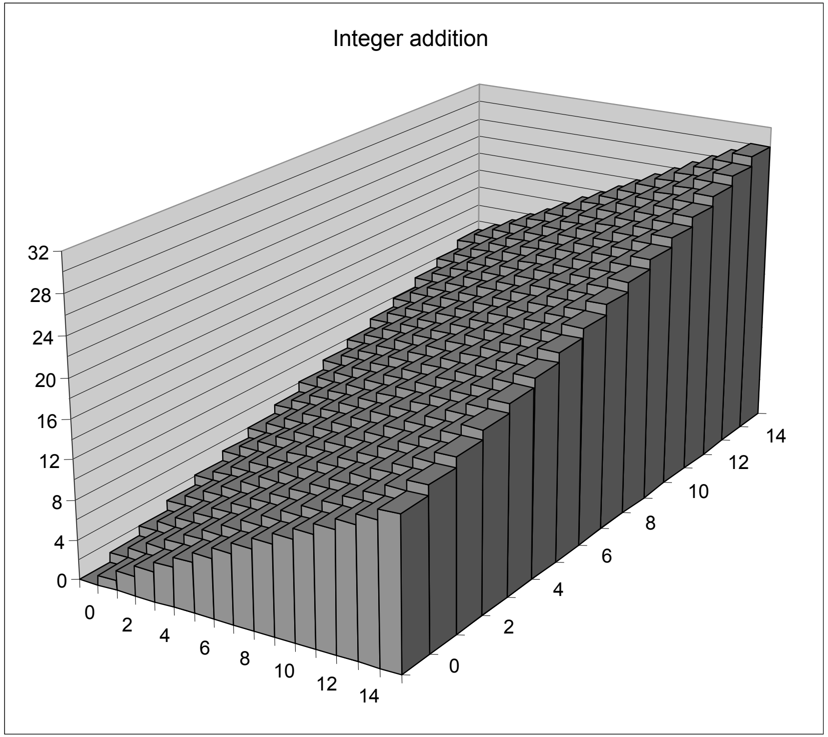


Figure: Image credit: CS:APP

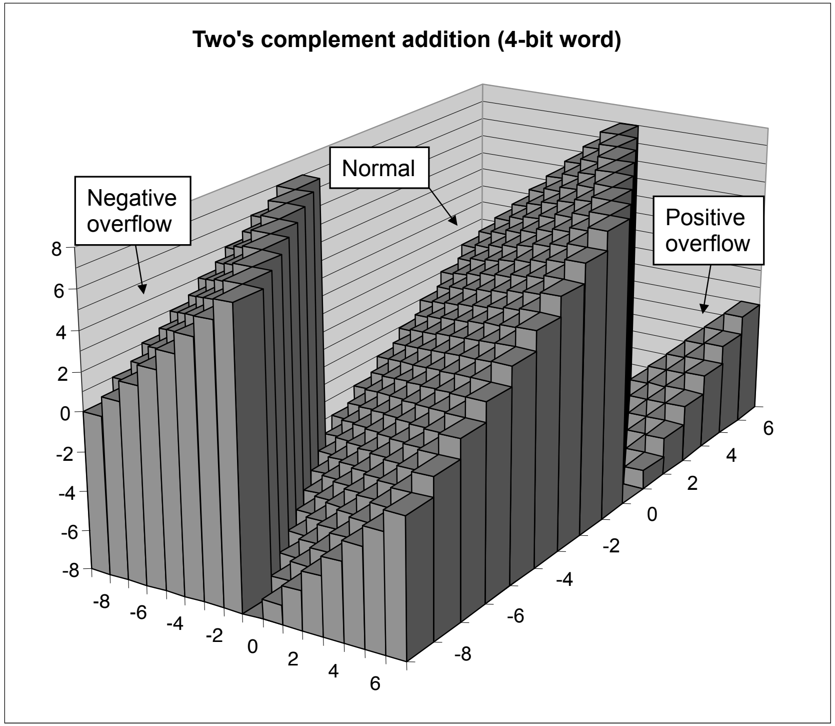


Figure: Image credit: CS:APP

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

# Table of contents

# Unsigned fixed-point binary for fractions

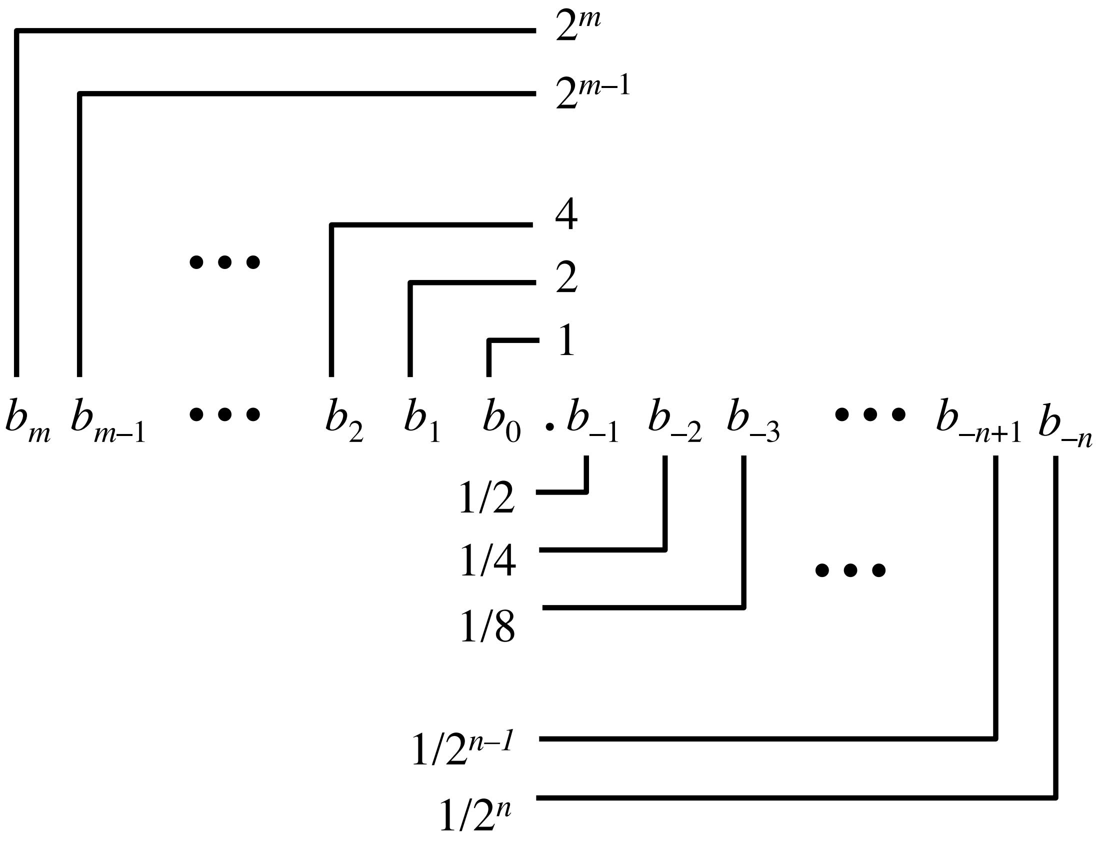


Figure: Fractional binary. Image credit CS:APP

# 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)

# 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

# Programming assignment 2: Graphs, trees, queues, hashes

## Programming Assignment 2 parts

1. edgelist: loading and printing a graph
2. isTree: needs either DFS (stack) or BFS (queue)
3. mst: a greedy algorithm
4. solveMaze: needs either DFS (stack) or BFS (queue)
5. findCycle: needs either DFS (stack) or BFS (queue)
6. hashTable: a separate chaining hash table

# Using `graphutils.h`

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

# Binary search tree

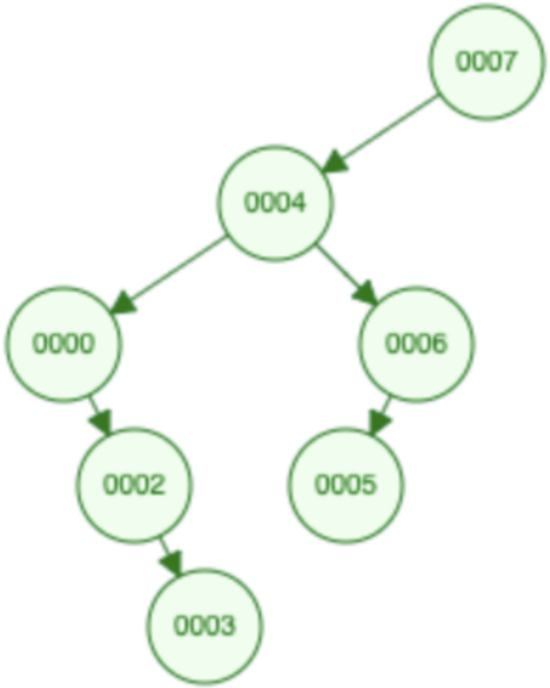


Figure: BST with input sequence 7, 4, 7, 0, 6, 5, 2, 3. Duplicates ignored.

# Binary search tree level order traversal

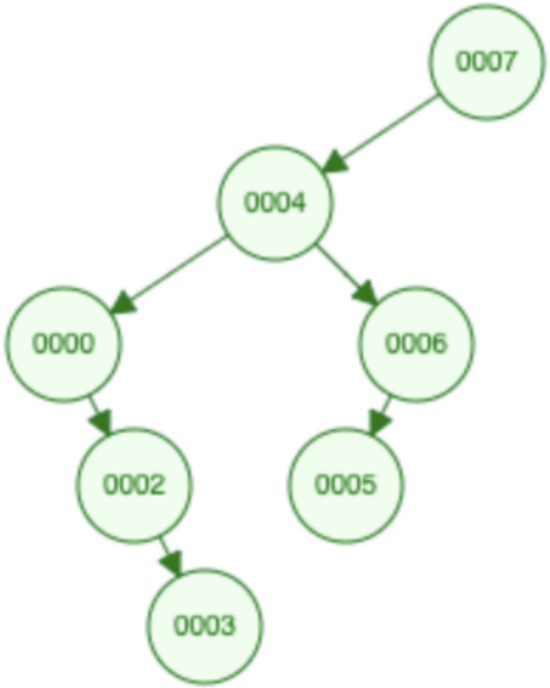


Figure: Level order, left-to-right traversal would return 7, 4, 0, 6, 2, 5, 3.

# 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).

# 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
};
```

# 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?