C Programming: structs, data structures

Yipeng Huang

Rutgers University

Feburary 4, 2021

Table of contents

Programming assignment

- Due in one week: 11:59pm Thursday, February 11.
- Find class's frequently asked questions on Piazza.
- Be careful not to disclose significant portions of your assignment code on Piazza.
- Goal today, Thursday: Work though examples of building a binary search tree and a queue using structs and pointers. Everything you need for part 4, balanced, and part 5, bstReverseOrder.

Looking ahead

Lecture plan

1. Tuesday, 2/9: Common mistakes in programming, debugging techniques.
2. Thursday, 2/11: Data representation of integers.
3. Tuesday, 2/16: Data representation of floating point numbers.

Reading assignment

- Computer Systems: A Programmer's Perspective Chapter 2.

Table of contents

Binary search tree

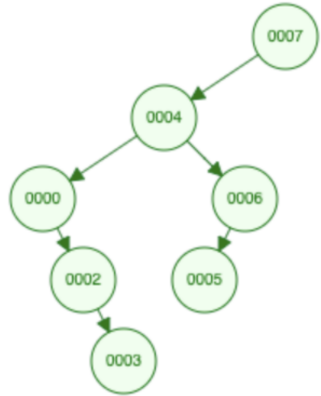

Figure: BST with input sequence 7, 4, 7, 0, 6, 5, 2, 3. Duplicates ignored.

Binary search tree level order traversal

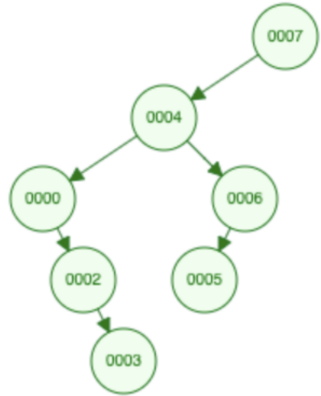

Figure: Level order, left-to-right traversal would return 7, 4, 0, 6, 2, 5, 3.

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).
- DEEP question: where is the stack in your recursive implementation in `bstReverseOrder.c`?

typedef

Why types are important

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

struct

arrays vs structs

- Arrays group data of the same type. The `[]` operator accesses array elements.
- Structs group data of different type. The `.` operator accesses struct elements.

These are equivalent; the latter is shorthand:

```
BSTNode* root;
```

- `(*root).key = key;`
- `root->key = key;`

When structs are passed to functions, they are passed BY VALUE.

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
};
```

Let's implement `insert()` and `delete()`

- Recursive implementations for `insert()` and `delete()`.
- Note the matching `malloc()` in `insert()` and `free()` in `delete()`.
- Tricky part: knowing what to pass as parameters and to return.
- Think: where should the data live, and how long should it persist?

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?

Tying it together in the `main()` function