Programming Technical Review Notes

Mike Micatka

August 18, 2016

Contents

1 Data Structures **1**
1.1 Linked Lists . . . 1
1.1.1 Description . . . 1
1.1.2 Pros . . . 2
1.1.3 Cons . . . 2
1.1.4 Implementation in C++ . . . 2
1.2 Binary Trees . . . 3
1.3 Tries . . . 3
1.4 Stacks . . . 3
1.5 Queues . . . 3
1.6 Vectors and Array Lists . . . 3
1.7 Hash Tables . . . 3

2 Algorithms **3**
2.0.1 Breadth-First Sort . . . 3
2.0.2 Depth-First Sort . . . 3
2.0.3 Binary Search . . . 3
2.0.4 Merge Sort . . . 3
2.0.5 Quick Sort . . . 3
2.0.6 Tree Insert/Find/etc. . . . 3

3 Concepts **3**
3.0.1 Bit Manipulation . . . 3
3.0.2 Memory (Stack vs Heap) . . . 3
3.0.3 Recursion . . . 3
3.0.4 Big-O Time . . . 3

4 Design Patterns **3**
4.0.1 Factory . . . 3
4.0.2 Singleton . . . 3

1 Data Structures

Comparison of common data structures:

	Linked List	Array	Dynamic Array	Balanced Tree	Hashed Array Table
Indexing					
Insert/Delete at Beginning					
Insert/Delete at End					
Insert/Delete at Middle					
Wasted Space (average)					

1.1 Linked Lists

1.1.1 Description

A linear collection of data elements (nodes) that contain contain a pointer to the next node. There are several sub-types of linked lists; singly-linked list, doubly-linked list, mulitiply-linked list, and circular-linked list among others.

1.1.2 Pros

- Dynamic, can be grown and pruned during run-time. Easy memory allocation/deallocation
- Node insertion and deletion are easily implemented
- Linear data structures are easily created using linked-lists

1.1.3 Cons

- Use more memory than arrays because of pointers needing to be stored
- Nodes must be read in order so are sequentially accessed
- Nodes are stored incontiguously so have longer access times
- Difficult to navigate backwards (doubly-linked lists are better but require an additional pointer)

1.1.4 Implementation in C++

Listing 1: C++ code using listings

```
struct Node
{
        Node* next;
        int data;
}

void insert_after(Node* cur_node, Node* new_node)
{
        new_node->next = cur_node->next;
        cur_node->next = new_node;
}
```

1.2 Binary Trees
1.3 Tries
1.4 Stacks
1.5 Queues
1.6 Vectors and Array Lists
1.7 Hash Tables
2 Algorithms
2.0.1 Breadth-First Sort
2.0.2 Depth-First Sort
2.0.3 Binary Search
2.0.4 Merge Sort
2.0.5 Quick Sort
2.0.6 Tree Insert/Find/etc.
3 Concepts
3.0.1 Bit Manipulation
3.0.2 Memory (Stack vs Heap)
3.0.3 Recursion
3.0.4 Big-O Time
4 Design Patterns
4.0.1 Factory
4.0.2 Singleton