

Data Structures Field Guide (v2)

Written by Krishna Parashar

Published by [The OCM](#) on February 10th, 2014

Contents

Why This Exists	5
Introduction	5
Linked Lists	6
Singularly Linked Lists	6
Invariants	6
Run Times	6
Code Example (Java)	6
Doubly Linked Lists	7
Invariants	7
Run Times	7
Code Example (Java)	7
Hash Tables	8
Invariants	8
Run Times	8
Algorithm	8
Methods	8
Stacks	10
Invariants	10
Code Example (Java)	10
Visual Example	10
Run Times	10
Queues	11
Invariants	11
Code Example (Java)	11
Visual Example	11
Run Times	11

Trees	12
Binary Trees	12
Invariants	12
Code Example (Java)	12
Visual Example	12
Run Times	12
2-3-4 Trees	13
Invariants	13
Code Example (Java)	13
Visual Example	13
Run Times	13
Red Black Trees	14
Invariants	14
Code Example (Java)	14
Visual Example	14
Run Times	14
Splay Trees	15
Invariants	15
Code Example (Java)	15
Visual Example	15
Run Times	15
Traversals	16
Invariants	16
Code Example (Java)	16
Visual Example	16
Run Times	16
Heaps	17

Graphs	18
Undirected Graphs	18
Invariants	18
Code Example (Java)	18
Visual Example	18
Run Times	18
Directed Graphs	19
Invariants	19
Code Example (Java)	19
Visual Example	19
Run Times	19
Breadth First Search	20
Depth First Search	20
Disjoint Sets	21
Colophon	22

Why This Exists

While taking my Data Structures class I found it very difficult to conceptually understand and network the plethora of new found concepts. Thus I wrote up this brief synopsis of the concepts I found useful to understanding the core ideas. This is of course by no means **comprehensive** but I do hope it will provide you with a somewhat better understanding computer architecture. Please feel free to [email me](#) if you have an questions, suggestions, or corrections. Thanks and enjoy!

Introduction

Linked Lists

Singularly Linked Lists

Invariants

1. An DList's size variable is always correct.
2. There is always a tail node whose next reference is null.

Run Times

- Insertion: $O(1)$
- Deletion: $O(1)$
- Indexing: $O(n)$
- Searching: $O(n)$

Code Example (Java)

```
public class SList {
    private SListNode head;           // First node in list.
    private int size;                 // Number of items in list.

    public SList() {                  // Here's how to represent an empty list.
        head = null;
        size = 0;
    }

    public void insertFront(Object item) {
        head = new SListNode(item, head);
        size++;
    }
}
```

Doubly Linked Lists

Invariants

1. An DList's size variable is always correct.
2. There is always a tail node whose next reference is null.
3. *item.next* and *item.previous* either points to an item or null (if tail or head).

Run Times

- Insertion: $O(1)$
- Deletion: $O(1)$
- Indexing: $O(n)$
- Searching: $O(n)$

Code Example (Java)

```
public class DList {
    private DListNode head;           // First node in list.
    private int size;                 // Number of items in list.

    public DList() {                  // Here's how to represent an empty list.
        head = tail = null;
        size = 0;
    }
    public void insertFront(Object item) {
        head = new SListNode(item, head);
        size++;
    }
    public void insertBack(Object item) {
        tail = new SListNode(item, head);
        size++;
    }
}
```

Hash Tables

Invariants

- n is the number of keys (words).
- N buckets exists in a table.
- $n \leq N \ll \text{possible keys}$.
- Load Factor is $n/N < 1$ (Around 0.75 is ideal).

Run Times

- Best: $O(1)$
- Worst: $O(n)$
- Resizing: Average $O(1)$ (Amortized)

Algorithm

Compression Function

- $h(\text{hashCode}) = \text{hashCode} \bmod N$.
- Better: $h(\text{hashCode}) = ((a * (\text{hashCode}) + b) \bmod p) \bmod N$
- p is prime $\gg N$, a & b are arbitrary positive ints.
- Really Large Prime: 16908799
- If $\text{hashCode} \% N$ is negative, add N .

Methods

insert(key, value):

1. Compute the key's hash code and compress it to determine the entry's bucket.
2. Insert the entry (key and value together) into that bucket's list.

find(key):

1. Hash the key to determine its bucket.
2. Search the list for an entry with the given key.
3. If found, return the entry; else, return null.

remove(key):

1. Hash the key to determine its bucket.
 2. Search the list for an entry with the given key.
 3. Remove it from the list if found.
 4. Return the entry or null.
-

Stacks

Invariants

Code Example (Java)

Visual Example

Run Times

Queues

Invariants

Code Example (Java)

Visual Example

Run Times

Trees

Binary Trees

Invariants

Code Example (Java)

Visual Example

Run Times

2-3-4 Trees

Invariants

Code Example (Java)

Visual Example

Run Times

Red Black Trees

Invariants

Code Example (Java)

Visual Example

Run Times

Splay Trees

Invariants

Code Example (Java)

Visual Example

Run Times

Traversals

Invariants

Code Example (Java)

Visual Example

Run Times

Heaps

Graphs

Undirected Graphs

Invariants

Code Example (Java)

Visual Example

Run Times

Directed Graphs

Invariants

Code Example (Java)

Visual Example

Run Times

Breadth First Search

Depth First Search

Disjoint Sets

Colophon

Written by [Krishna Parashar](#) in Markdown on Byword. Used [Pandoc](#) to convert from Markdown to Latex.