Data Structures Field Guide (v2)

Written by Krishna Parashar

Published by The OCM on February 10th, 2014

Contents

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)

```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)

```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 << $ 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(hashCode) = hashCode mod N.
- Better: h(hashCode) = ((a * (hashCode) + b) mod p) mod N
- p is prime >> N, a & b are arbitrary positive ints.
- Really Large Prime: 16908799
- If 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.