Algorithms and Data Structures

Notes on Algorithms, Data Structures and other concepts

ERIC LI

February 9, 2019

Contents

1 Disclaimer

The author(s) of this document assume(s) no responsibility or liability for any errors or omissions in the content of this document. The information contained in this site is provided on an “as is” basis with no guarantees of completeness, accuracy, usefulness or timeliness or of the results obtained from the use of this information.

Information provided in this document has been taken from various sources and is by no means a comprehensive record of the source’s views nor of the concepts represented.

2 Trees

2.1 Structure

2.1.1 Terminology

Node

3 Linked Lists

Linked lists are helpful in addressing some of the limitations that arrays face, such as inherent dynamic memory allocation and ease of modification. Instead of memory being assigned for use (as in the programming language C) linked lists are built on the premise of allocating memory as needed, and inserting or deleting as required by the user or data set.

3.1 Structure

A linked list is a linear data structure constructed from building blocks called nodes. Like other data structures, a node contains two items: data, which can be simple, compound, pointer, etc. and a memory address reference (or link to the next node). The address reference will access the next node’s memory address in the list, and creates a ‘chain-like’ relationship which will naturally produce a linear structure. There are two important nodes that are called the ‘head node’ and the ‘tail node’, which will influence operations and be explained in depth in the following sections.

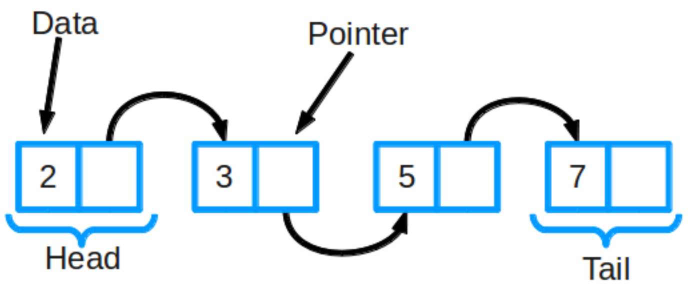

Figure 1: Linked List

3.2 Operations

Linked lists support basic ADT operations of inserting, deleting, and modifying.
More specifically, linked lists are constructed by inserting or deleting additional nodes at specified locations in the list.
Insertion or deletion at the beginning or end of a list is appropriately named ‘Head Insertion’ and ‘Tail Insertion’ and, ‘Head Deletion’ and ‘Tail Deletion’.

There are two important nodes:

- Head Node
  - Head insertion is much quicker due to the lack of list traversing done
- Tail Node
  - Has an address reference of NULL
  - Entire list must be traversed before tail insertion can occur

3.3 Basic Implementation

1. Define a node

```
    typedef struct node{
        int num; // data
        struct node *ref; // address
    } node;
```

2. Dynamic creation of nodes

```
    node *new_node(void){
        node *p=NULL;
        p=(node *)calloc(1,sizeof(node));
        p->num=0;
        p->ref=NULL;
        return p;
    }
```

4 Queues

Queues are built heavily on concepts from linked lists and naturally, follows the same linear data structure. Queues provide a similar environment for "tail insertion" and "head deletion", which are appropriately named "enqueue" and "dequeue" respectively.

5 Sources

https://www.cs.cmu.edu/ adamchik/15-121/lectures/Linked
https://medium.com/@kenny.lin/singly-linked-lists-5cfdec60bea0