CS 225
a.k.a.
Data Structures

Spring 2011
University of Illinos
as taught by Cynda Heer

notes by Michele R. Esposito

Please, let report any error or type-o at
michekeresposito@gmail.com

Contents

1 Linear Data Structures

1.1 Linked Lists

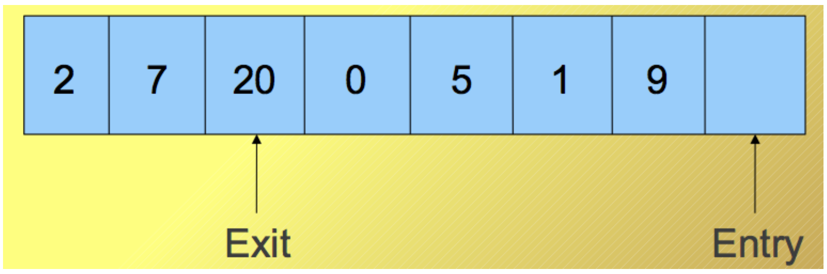
Operations	**Running Time**	**Notes**
Insert front	O(1)	
Insert back	O(n)	O(1) in DLL w Sentinel node
Insert random	O(n)	
Delete random	O(n)	O(1) in double linked list
Lookup	O(n)	

Linked lists are a very simple data structure, which is a building block for many other data structures, like trees. There are two common types of Linked lists: **Singly linked(SLL)** and **double linked (DLL)**. SLL allows you to traverse only in one direction, whereas DLL allows you to traverse both ways.
A **Sentinel Node** is an empty node that is used to keep track of a certain location inside a linked list. They are usually positioned at the beginning and at the end of the list in order to speed up insertion and removal.
Disadvantages: Linked list are usually very slow, as they are not stored contiguously in memory. In order to access the elements, you need to dereference all the pointers, which can be very slow.
Advantages: Its useful to use linked list if you know that you will be moving your data around a lot, as its easy to rearrange them, without needing additional space. There are some versions of Malloc which are implemented using linked lists.

1.2 Stack and Queues

Operations	**Running Time**	**Notes**
Pop	O(1)	Dequeue in queues
Push	O(1)	Enqueue in queues

Stack and queues are two simple, yet very useful, data structures. Stacks are FILO, First In Last Out data structures, whereas queues are FIFO, First In/-First Out. Both stacks and queue are usually implemented either using arrays, or using linked lists. Either way, a constant runtime is guaranteed for the above operations.
Implementation: Implementing both a stack and a queue using a Linked List(LL) is straight forward. In the stack, you only need to keep of the top element, which is easy to do if you use a sentinel node. In a queue, you need to keep track of the first and last node, again easy to do if you use sentinel nodes. Arrays are slightly more complicated, as you have the problem of filling up the array. Typically, when you fill up the array, you double the size and then copy the elements over. This operation will still give you an amortized time of $O(1)$ per insert. In a stack, you simply need to keep track of the position of the top element, then resize if the length of the array is equal to the index of the top element.
If you want to implement a queue using an array, then you need to keep track of the entry point and the exit point.



However, this time you resize only if $abs(exitPoint - entryPoint) >= n$ where $entryPoint$, $exitPoint$ are indexes, and n is the length of the array. This is because your indexes can cirulate thought the array.
Notes: You can actually implement a queue using two stacks. The trick, is using one of the stacks enqueue elements, and the other to dequeue. Whenever you want to dequeue an element, simply reverse every other element inside the other stack, and you will be able to dequeue elements from there.

2 Trees

A tree is one of the most used data structures in CS, and its indeed very useful. There are many different kinds of tree, for all sort of purposes. In here, I will be covering some of the most common ones.

2.1 Binary Search Tree

Operations	**Average Case**	**Worst case**
Insert	O(log n)	O(n)
Removal	O(log n)	O(n)
Lookup	O(log n)	O(n)

A binary search tree (BST) is one of the simplest basic trees. It is defined in this way:
$T = \{\}$ or
$T = \{r, T_r, T_l\}$ where
$\forall\ keys \in T_r \geq key(r)$ and $\forall\ keys \in T_l < key(r)$.
Advantages: BST are straight forward to implement, and have a good average case $O(logn)$. They are good if you already know the median of the data that you are going to insert inside the tree. Like that, you can use the median value as root for your tree. Otherwise, you will run into an imbalanced tree.
Disadvantages: BST do not guarantee height $\log n$. Say, for example, if you were going to insert elements from 1 to 10 in this order. Then, you would get a tree of height 10, which is bad news, because its just a fancy linked list. Thus, *the height is dependent on insert order.*