Exam

Binary Search Tree

The nodes in BST stay sorted that

All values in the left subtree must be less than or equal to the root node

All values in the right subtree must be greater than the root node

The problem in BST

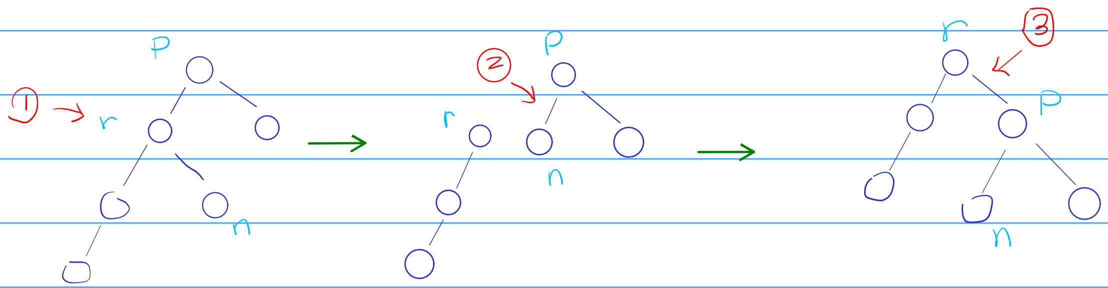
All nodes have one subtree and each search has go from root down to leave node to find out the search item does not exist

AVL Tree

self balancing binary tree

the height of two child subtrees of any node differ by at most one

Add and Remove may require the tree to be rebalanced by one or more tree rotations



rotateRight (p)

1 tmp = p lchild // tmp = r

2 p lchild = r rchild // p lchild = n

3 r rchild = p // r rchild = p

return p

String Operation Terminology

size() : return the number of characters of the string

empty() : return true if the string is empty and false otherwise

operator [i] : return the character at index i of the string (no array bounds checking)

at(i) : return the character at index i of the string. An out of range exception is thrown if i is out of bounds

insert(i, B) : insert string B prior to index i in the string A and return a reference to the result

append(B) : append string B to the end of string A and return a reference to the result

erase(i, m) : remove m number of characters starting at index i and return a reference to the result

substr(i, m) : return the substring of string a of length m starting at index i

find(B) : if string B is a substring of string A, return the index of the beginning of the first occurrence of B in A, else return the length of A

c_str() : return a C-style string containing the contents of string A

Dynamic Programming

an algorithmic technique for solving an optimization problem by breaking it down into simpler subproblems and ulitizing the fact that the optimal solution to the overall problem depends upon the optimal solution to its subproblems

an algorithmic techique for solving an optimization problem by breaking it down into smaller subproblems

often these subproblems overlap in some way

Longest common subsequence

Defined as the longest subsequence that is common to all the given sequences for which the elements of the subsequence are not required to occupy consecutive positions within the original sequences.

Set ADT

a collection of distinct objects

No duplicate elements in a set

No explicit notion of keys or even an order

If the elements in a set are comparable, then we can maintain sets to be ordered

Text compression

- efficiently encode string A into a small binary string B (using only the characters 0 and 1)
- increase throughput
- begin by finding the frequency of each character in string A and order it in ascending order by frequency
- build a binary tree by combining the frequency minimums

Merge Sort

- each node represents a recursive call of merge sort and stores
- the root is initial call
- the leaves are calls on subsequences of size 0 or 1

Quick Sort

- a randomized sorting algorithm based on the divide-and-conquer paradigm
- pick a random element x called pivot and partition array A into
  - L elements less than x
  - E element equal to x
  - G elements greater than x
- recursive: sort L and G
- conquer: join L, E, and G

Divide and conquer

- a general algorithm design paradigm
- Divide : divide the input data S in two disjoint subset S1 and S2
- Recur : solve the subproblems associated with S1 and S2
- conquer : combine the solutions for S1 and S2 into a solution for S

Merge sort

- a sorting algorithm based on the divide-and-conquer paradigm
- like heap sort
  - uses a comparator
- unlike heap sort
  - accesses data in a sequential manner
- Divide : partition S into two sequences S_1 and S_2 of about $n/2$ element each
- Recur : recursively sort S_1 and S_2
- conquer : merge S_1 and S_2 into an unique sorted sequence

Radix - sort

is a specialization of lexicographic-sort that uses bucket sort as the stable sorting algorithm in each dimension

is applicable to tuples where the keys in each dimension i are integers in the range [0, N-1]

Trie Data Structure

a type of k-ary tree used for locating specific keys from within a set

These keys are most often strings with links between each nodes defined by individual character

Quick sort

10	80	30	90	40	50	70

↑ pivot

i = index of smaller element = -1

j = loop variable = 0

10 < 70 T ++i

swap(array[i], array[j])

++j

10	80	30	90	40	50	70

i = 0 , j = 1

↑ pivot

80 < 70 F pass ++j

i = 0 , j = 2

30 < 70 T ++i

swap(array[i], array[j])

++j

| 10 | 30 | 80 | 90 | 40 | 50 | 70 |

i = 1 , j = 3

pivot

90 < 70 F pass ++j

i = 1 , j = 4

40 < 70 T ++i

swap(array[i], array[j])

++j

| 10 | 30 | 40 | 90 | 80 | 50 | 70 |

i = 2 , j = 5

50 < 70 T ++i

swap(array[i], array[j])

++j

| 10 | 30 | 40 | 50 | 80 | 90 | 70 |

i = 3 , j = 6

j = 6 , break loop

swap(array[i+1], array[P])

| 10 | 30 | 40 | 50 | 70 | 90 | 80 |

pivot

repeat

go back to 80

| 10 | 30 | 40 | 50 | 70 | 90 | 80 |

$\Uparrow$ pivot

i = 4
j = 5

swap(array[i+1], array[p])

Merge sort

| 38 | 27 | 43 | 3 | 9 | 82 | 10 |

if l > r
divide in half

| 38 | 27 | 43 | 3 | | 9 | 82 | 10 |

if l > r
divide in half

if l > r
divide in half

| 38 | 27 | | 43 | 3 | | 9 | 82 | | 10 |

if l > r
divide in half

if l > r
divide in half

if l > r
divide in half

| 38 | | 27 | | 43 | | 3 | | 9 | | 82 |

Start swap

| 27 | | 38 | | 3 | | 43 | | 9 | | 82 | | 10 |

| 27 | 38 | | 3 | 43 | | 9 | 82 | | 10 |

1 2 3 1 2 3 1 2 3 1 2

| 3 | 27 | 38 | 43 | | 9 | 10 | 82 |

1 2 3 4 5 6 1 2 3 4 5 6 7

| 3 | 9 | 10 | 27 | 38 | 43 | 82 |

Bucket Sort

7, d → 1, c → 3, a → 7, g → 3, b → 7, e

Phase 1

0	1	2	3	4	5	6	7	8	9
	↑ 2		↑ 3				↑ 1		
	1, c		3, a				7, d		
			↑ 5				↑ 4		
			3, b				7, g		
							↑ 6		
							7, e		

Phase 2

1, c → 3, a → 3, b → 7, d → 7, g → 7, e

Bucket Sort

The keys are used indices into an array and cannot be arbitrary objects

No external comparator

Stable Sort Property

The relative order of any two items with the same key is preserved

Radix Sort

1001		0010		1001		1001		0001
0010		1110		1101		0001		0010
1101	→	1001	→	0001	→	0010	→	1001
0001		1101		0010		1101		1101
1110		0001		1110		1110		1110

170	45	75	90	802	24	2	66
170	90	802	2	24	45	75	66
802	2	24	45	66	170	75	90
2	24	45	66	75	90	170	802