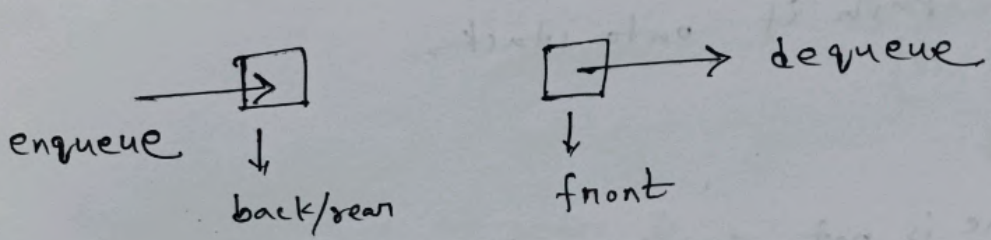
Queue

back/rear → add to one end

front → remove from other end

FIFO



1. enqueue
2. dequeue
3. peek → returns the item that is at the front of the queue.

Applications:

Palindrome: radar
31413

begin with an empty queue, an empty stack,
    input string

for each char in input do
    if char is a letter
        enqueue it into queue
        push it onto stack

end for

while queue is not empty, do:
    c_1 = dequeue the queue
    c_2 = pop the stack
    if c_1 != c_2
        return false

end while

return true

implementation:

Array based

- new item being enqueued at the end of the array (~~cost~~ constant cost)
- dequeue the ~~the~~ item in the front of the queue from beginning of the queue (linear cost) → [shifting all item to fill hole]

front item is at index 0

rear/back is at (size - 1)

```
class Queue:
    def __init__(self):          , capacity
        self.size = 0
        self.data = [none] * capacity
        self.front = 0
```

front

rear

✓5, 7, 9, 15, -4

✓7, 9, 15, -4 ← dequeue

9, 15, -4 ← dequeue

front

rear

In circular array, front is not at index 0

If front is the index of front item,

index of rear item = (front + size − 1) % capacity

idx of next available slot = (front + size) % capacity

Length of array

0	1	2	3	4	5	6	7	8
10,	20,	30,	40,	50	-	-	-	-
f				r				

20, 30, 40, 50 ← dequeue
f r

30, 40, 50
f r

40, 50
f r

50
f=r

enqueue 5, 7, 9, 15, -4, 17

0	1	2	3	4	5	6	7	8
-4	17	-	-	-5	7	9	15	
	r			f				

```
int j = front
for i in range(size):
    print(data[j])
    j = (j+1) % len(data)
```

forward iteration

front to rear

backward (rear to front)

```
j = (front + size - 1) % len(data)   ← (idx of rear)
for i in range(size-1, -1, -1):
    print(data[j])
    j = j - 1
    if j == -1:
        j = len(data) - 1
```

Circular array use করা হয় যখন shift করতে না হয় dequeue করার সময়।

We just have to maintain the position/idx of front item.

resize cyclic array

0	1	2	3	4	5	6	7	8	9
-4	17				5	7	9	15	
	n				f				

resize ⟶ capacity 11

0	1	2	3	4	5	6	7	8	9	10
5	7	9	15	-4	17 ✓					
f					n					

Linked List based

⟶ add new item being added at end of the array, using tail reference to append at constant cost.

⟶ remove front item from beginning of the list at constant cost.

29 Aug 2022, 13:36

Priority Queue

token numbers
↓
secondary key
(underlying queue)

arrival order
↓
actual queue

Priority may change over time
solely arrival order এর জন্য
depend করে না।

lower values for the key means that object has higher priority, so will be removed earlier.

sort([] a) → non-decreasing order (বা increasing order)

```
PQ pq = new PQ()
for i in range (len(a)):
    pq.add(a[i], a[i])
        ↓ item   ↓ key
for i in range (len(a)):
    a[i] = PQ.remove()
```

if we want to sort in non-increasing order

Two choices!

a) lower key present lower priority

b) sort first, and then reverse array or

c) Add objects from back

```
for i in range(len(a)-1, -1, -1):
    a[i] = pq.remove()
```

Implementation

Unsorted/unordered array

- adding item is independent of number of items in PQ
  - takes constant time
  - add it ~~end~~ next available position

remove → searching at most n times in PQ for lowest key, shift n-1 times to left to fill up hole

29 Aug 2022, 13:36

→ remove

- binary search: $\log_2(n)$ comparison + $(n-1)$ shift
- selection sort: n " + $(n-1)$ "

adding: 1 (constant)

Sorted array (non-increasing order) → ...

adding → insertion sort

n comparisons + n shifts + 1 (add)

remove → 1 (constant)

↳ lowest key is always in last used slot of array. so remove takes constant cost.

Searching :-

- Sequential Search through a sequence
- Binary search through a sorted random-access sequence

Advantage of sequential search

- data sort হতে হবে না
- random access container লাগবে না

Sequential Search → (n number of comparison)

Array :

```
def seqSearch( data[], size, key):
    for i in range(size):
        if key == data[i]
            return i

    return -1
```

29 Aug 2022, 13:36

Linked List

```
def seqSearch(head, key):
    n = head
    i = 0
    while(n != null):
        if key == n.value:
            return i
        n = n.next
        i += 1
    return -1
```

Binary Search → [$\log_2(n)$ comparison]

(data must be sorted in non-decreasing order)

```
def binSearch(data[], size, key):
    l = 0
    r = size - 1
    while(l <= r):
        mid = (l + r)/2
        if key == data[mid]:
            return mid
        elif key > data[mid]
            l = mid + 1
        else:
            r = mid - 1
    return -1
```

after k steps,

we have an array of size $\frac{n}{2^k}$

# of comparison = 1 in all steps

step	Array size
0	$n/2^0 = n$
1	$n/2^1 = \frac{n}{2}$
2	$n/2^2 = \frac{n}{4}$
⋮	⋮
k	$n/2^k = \frac{n}{2^k}$

$$\frac{n}{2^k} = 1$$

$$2^k = n$$

$$k = \log_2(n) \leftarrow (\text{steps})$$

∴ total number of comparison = $\log_2(n) \times 1$

$= \log_2(n)$

29 Aug 2022, 13:37

Sorting

- bubble
- selection
- Insertion

Selection sort

input: 17 3 9 21 2 7 5

n = 7

step1: | 17 3 9 21 2 7 5 — min is 2 exchange with 17

step2: 2 | 3 9 21 17 7 5 — min is 3 no exchange

step3: 2 3 | 9 21 17 7 5

step4: 2 3 5 | 21 17 7 9 — min is 5 swap with 9

step5: 2 3 5 7 | 17 21 9

step6: 2 3 5 7 9 | 21 17

step7: 2 3 5 7 9 17 | 21

Array based

```
def selectionSort(data[]):
    for i in range(len(data)-1):
        minIdx = i
        for j in range(i+1, len(data)):
            if data[j] < data[minIdx]:
                minIdx = j
                temp = data[j]
        if i != minIdx:
            temp = data[i]
            data[i] = data[minIdx]
            data[minIdx] = temp
```

Linked List based

```
def selectionSort(head):
    if head == null || head.next == null
        return
    n = head
    while(n != null):
        minNode = n
        p = n.next
        while (p != null):
            if p.value < minNode.value:
                minNode = p
            p = p.next
        if n != minNode:
            temp = n.value
            n.value = minNode.value
            minNode.value = temp
        n = n.next
```


Step	Array Size	Max number of Comparisons
1	n	n-1
2	n-1	n-2
3	n-2	n-3
⋮	⋮	⋮
n-1	2	1

∴ total number of comparison = $1+2+3+\dots+(n-1)$

$$= \frac{(n-1)\times n}{2}$$

$$= \frac{n^2-n}{2}$$

$$\simeq n^2$$

best & worst cases are same

Sort এলগরিদম n^2 comparison

selection sort

weakness

Not Adaptive

29 Aug 2022, 13:37

Insertion Sort:

input: 17 3 9 21 2 7 5

n = 7

Step 1: 17 | 3 9 21 2 7 5 (move 3 to left)

2: 3 17 | 9 21 2 7 5 (move 9 to left)

3. 3 9 17 | 21 2 7 5 (move 21 to left)

4. 3 9 17 21 | 2 7 5 (move 2)

5. 2 3 9 17 21 | 7 5 (move 7)

6. 2 3 7 9 17 21 | 5 (move 5)

7. 2 3 5 7 9 17 21 (stop)

Array → doubly linked list

```
def insertionSort(data[]):
    // i denotes where partition is
    for i in range(1, len(data)):
        key = data[i]
        j = i - 1   // use j to scan left to insert key
        while j >= 0 && data[j] > key:
            // shift item right to make room
            data[j+1] = data[j]
            j = -1
        // found position where key can be inserted
        data[j+1] = key
```

Best case : $n-1$ (already sorted array)

Wirst case : n^2 ⟶ (reversed order & sorted air)

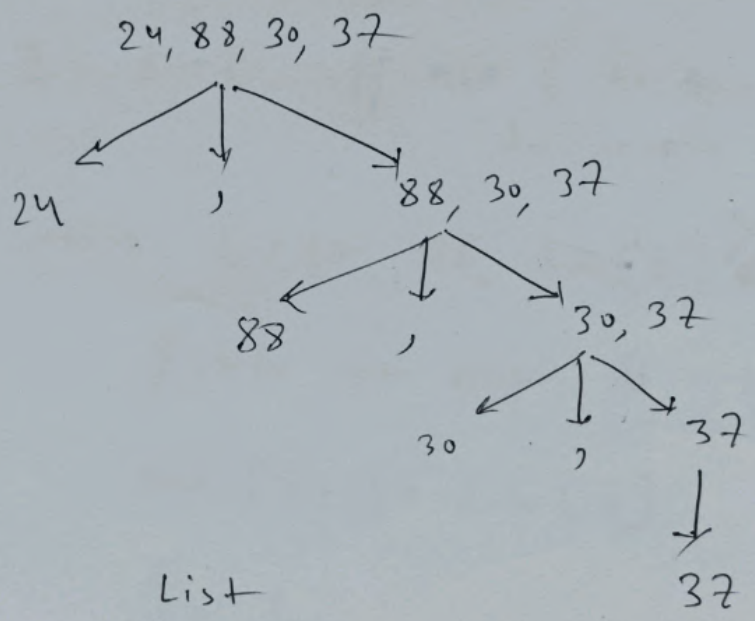
Recursion

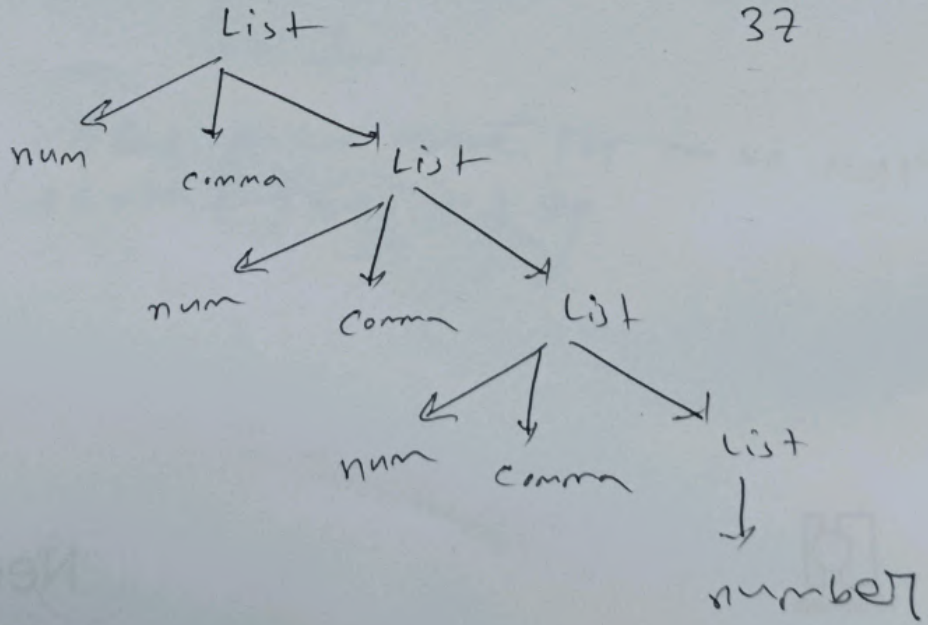
- overlapping subproblem
- Optimal substructure

1. A List of numbers

<24, 88, 30, 37>

Recursion tree diagram





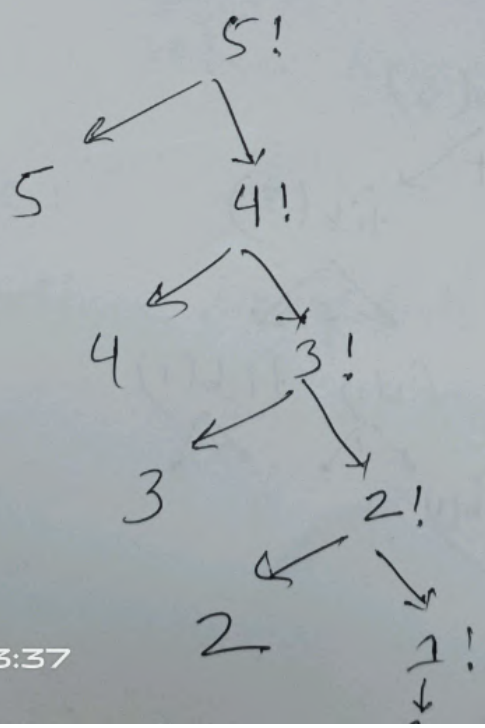
29 Aug 2022, 13:37

base case/stopping ~~or~~ condition: A List is either a number

#2. Factorial

```
def recur(n):
    if n == 0 || n == 1:
        return 1
    elif n > 1: else:
        return n x (n-1)!
```

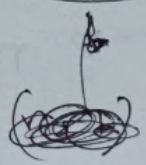
n x recur(n-1)

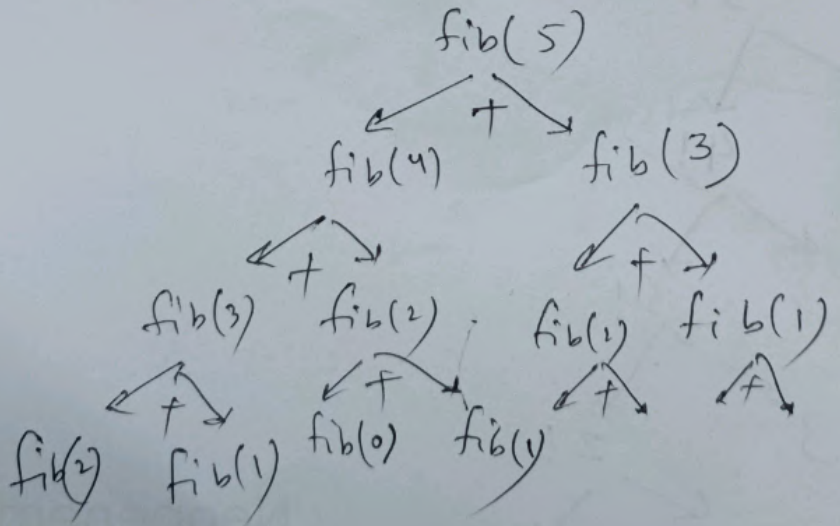


3. Fibonacci (Redundancy in computation → inefficiency)

```
def fibo(n):
    if n == 0:
        return 0
    elif n == 1
        return 1
    else:
        return fibo(n-1) + fibo(n-2)
```

→ overcome ~~করে~~ ✍: Memorization topic ~~করবো~~।





→ iterativeSum

→ RecursivelySum

↓

if k is the only element
sum = k

otherwise

k + sum(n.next)

```
def recursiveSum(linked_list):
    if linked_list.size() == 1:
        return linked_list.head.value
    else:
        return linked_list.head.value +
               recursiveSum(linked_list.next)
```

Examples:

1. Length of a string.

```
def strLength(string):
    if string == "":
        return 0
    else:
        return 1 + strLength(string.substring(1))
```

if not string:
    return 0

string[1:]

2. Length of a linked list

```
def listLength(list):
    if list == null:
        return 0
    else
        return 1 + listLength(list.next)
```

head

3. Sequential Search (Linked List)

a)

```
def contains(head, k):
    if head == null:
        return false
    elif head.item == k:
        return true
    else
        return contains(head.next, k)
```

b) Seq. Search (Array):

```
def contains(arr[], l, k):
    if l >= len(arr):
        return false ⟶ (-1)
    elif a[l] == k:
        return true ⟶ (l)
    else:
        return contains(arr, l+1, k)
```

1. Binary Search in a sorted array

↓

→ random access

মাঝে হবে so

Array must.

```
def contains(arr[], l, r, k):
    if l > r:
        return false  ⟶ (-1)

    else:
        mid = (l+r)/2
        if k == arr[mid]
            return true  ⟶ (mid)

        elif k > arr[mid]:
            return contains(arr, mid+1, r, k)

        else
            return contain(arr, l, mid-1, k)
```

5. a) Finding max in a sequence (Linear version)

Linked list based ✓

```
def findMax(head):
    if head.next == null:
        return head.value
    else
        maxRest = findMax(head.next)
        return math.max(head.value, maxRest)
```

b) Array based ✓

```
def findmax(arr[], l, r)        # r: key
    if l == r:                  # r: key
        return a[l]
    else:
        maxRest = findMax(arr, l+1, r)
        return math.max(arr[l], maxRest)
```

findMax(arr, 0, len(arr)-1)

6. Find max in array (binary version)

```
def findMax(arr[], l, r):
    if l == r:
        return arr[l]
    else
        mid = (l+r)/2
        maxLeftHalf = findMax(arr, l, mid-1)
        maxRightHalf = findMax(arr, mid+1, r)
        return math.max(maxLeftHalf, maxRightHalf)
```

max+1

7. Selection Sort

a) (Linked List)

```
def selectSort(head):
    if head == null || head.next == null:
        return;
    else
        minNode = findMinNode(head)
        swap(head, minNode)
        selectSort(head.next)

def findMinNode(head):
    if head.next == null:
        return head
    else
        minNode = head
        minNodeRest = findMinNode(head.next)
        if minNodeRest.value < head.value:
            minNode = minNodeRest
        return minNode
```

b) array

```
def selectSort(arr[], l, r):
    if l >= r:
        return;
    else:
        minIdx = findMinIdx(arr, l, r)
        swap(arr, l, minIdx)
        selectSort(arr, l+1, r) #
```

```
def findMinIdx(arr[], l, r):
    if l == r:
        return l;
    else:
        mid = (l+r)/2
        minIdxLeft = findMinIdx(arr, l, mid-1)
        minIdxRight = findMinIdx(arr, mid+1, r)
        minIdx = minIdxLeft
        if arr[minIdxRight] < arr[minIdxLeft]
            minIdx = minIdxRight
        return minIdx
```

29 Aug 2022 13:38

8. Insertion Sort

```
def insertSort(arr[], l, r):
    if l >= r:
        return;
    else
        insertionSort(arr, l, r-1)
        key = arr[r]
        j = r - 1
        while j >= 0 && key < arr[j]:
            arr[j+1] = arr[j]
            j -= 1
        arr[j+1] = key
```

```
insertSort(arr, 0, len(arr) - 1)
```

9. Sum of integers through 1 to N

```
def sumN(n):
    if n==1:
        return 1
    else
        return n + sumN(n-1)
```

10. Exponentiation $\longrightarrow a^n$

2^3

$2^{2} \times 2^{1}$

$2^{1} \times 2^{1} \times 2^{1}$

```
def exp(a, n):
    if n==0:
        return 1
    else:
        return a * exp(a, n-1)
```

efficient:

```
if n == 0:
    return 1
elif n % 2 == 0:
    return exp(a, n/2) * exp(a, n/2)
else:
    return exp(a, (n-1)/2) * exp(a, (n-1)/2) * a
```

but it is **also redundant**

huge boost হবে modify করলে

modified version

```
def exp(a, n):
    if n == 0:
        return 1
    elif n % 2 == 0:
        temp = exp(a, n/2)
        return temp * temp
    else:
        temp = exp(a, n-1/2)
        return temp * temp * a
```

→ temporary variable

Simple case of memorization

Memoization

```
def fib(n):
    F = [-1] * (n+1)
    return M_fib(n, F)

def M_fib(n, F):
    if n < 2:
        return n
    else:
        if F(n) == -1:
            F[n] = M_fib(n-1) + M_fib(n-2)

    return F[n]
```

F কে global variable হিসাবে use করি যাতে প্রতিবার function এ pass করতে না হয়।

Space overhead লাগে extra

Dynamic Programming

Iterative

```
def fib(n):
    F = [none] * (n+1)
    F[0] = 0
    F[1] = 1

    for i in range(2, n+1):
        F[i] = F[i-1] + F[i-2]

    return F[n]
```

this avoids the overhead of recursion by using iteration, so tends to run faster

Space requirement ~~no~~ update

(no need f array of n+1 capacity)

```
def fib(n):
    f_2 = 0   ⟶ n-2
    f_1 = 1   ⟶ n-1
    f = n
```

29 Aug 2022, 13:38

```
for i in range(2, n+1):
    f = f_1 + f_2
    f_2 = f_1
    f_1 = f
return f
```

Time complexity of recursive → 2^n (exponential)

iterative → O(n) (linear)

Space complexity of iterative → O(1)

(constant)

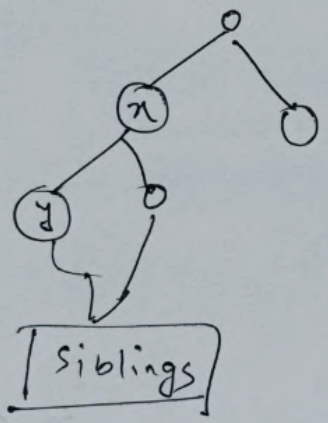
recursive → O(n)

linear

Trees

Forests are disjoint set of trees

Graph যদি acyclic, connected হয় → tree



x is above y
y is a descendant of x
(y below x)
x is an ancestor of y

leaves / terminal nodes → no children

M-ary tree → exactly m children

Binary tree → exactly 2 children (m=2)

→ external / sentinel nodes (leaf এর নিচে invisible nodes)
↳ no children

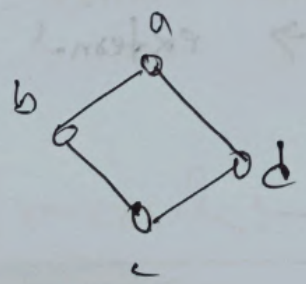
29 Aug 2022, 13:38

Binary Trees

```
class Node:
    def __init__(self, item, left, right, parent):
        self.item = item
        self.left = left
        self.right = right
        self.parent = parent
```

(item: object; left: node; right: node; parent: node)

```
n = Node(3, l, r)
n = n.left   (left subtree)
n = n.right  (right subtree)
```



a-b-c-d-a (cycle)

a-b-c-d (simple path)

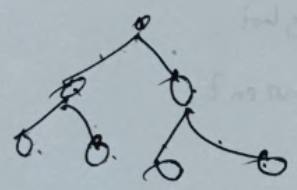
a-b-c ⟶ path

connected graph (a path between each pair of vertices)

Graph → tree

1. $n-1$ edges, no cycle
2. $n-1$ edge, connected
3. exactly one path between each pair of vertices
4. if any edge is removed → disconnected

binary tree



n → internal nodes

$n+1$ external nodes

$2n$ internal links/edges

$n-1$ → internal nodes

$n+1$ → external nodes

Level / Depth of node

$$level(n) = \begin{cases} 0, & \text{if } n \text{ is the root} \quad (parent(n) = null) \\ 1 + level(parent(n)), & \text{otherwise} \end{cases}$$

29 Aug 2022, 13:39

height of tree

↳ maximum of the levels of nodes

0 ⟶ height = 0

$$height(n) = \begin{cases} -1 & \text{if tree is empty } (n = null) \\ 1 + \max(height(n.left), height(n.right)) \end{cases}$$

height of binary tree with n internal nodes is at least $\log n$ and at most $n-1$

Tree Traversal :—

depth order

pre-order: root, left, right (depth first) (dfs)

in-order: left, root, right

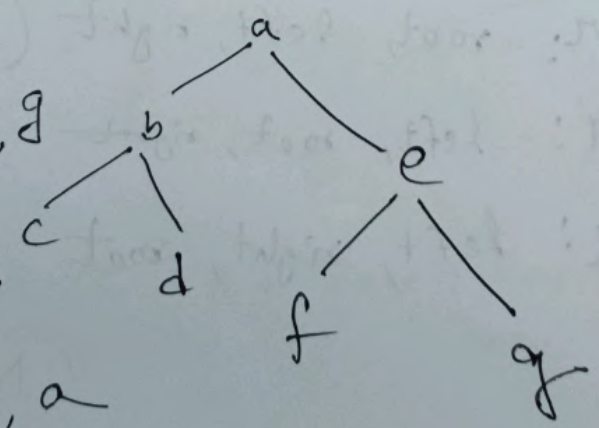
post-order: left, right, root

```
def preOrderVisit(node):
    if node == null:
        return
    else:
        visit(node)
        preOrderVisit(node.left)
        preOrderVisit(node.right)
```

test: → preOrderVisit(root)

Level Order Traversal : Breadth ~~first~~ (bfs)

Example of traversal →



pre: → a, b, c, d, e, f, g

in: → c, b, d, a, f, e, g

post: → c, d, b, f, g, e, a

29 Aug 2022, 13:39

Some recursive tree algo

counting the nodes

```
def count(node):
    if node == null:
        return 0
    else:
        return 1 + count(node.left) + count(node.right)
```

Copying a binary tree

```
def copy(node):
    if node == null:
        return null
    else:
        copy = Node(node.element, copy(node.left), copy(node.right))
        return copy
```

Comparison of two trees

```
def same(node1, node2):
    if node1 == null || node2 == null:
        if node1 == null && node2 == null:
            return true
        else
            return false
    else:
        return node1.element.equals(node2.element)
            && same(node1.left, node2.left)
            && same(node1.right, node2.right)
```

find Rightmost node 3 same

find Leftmost node

```
def findLeft(node):
    if node.left == null:
        return node
    else
        return findLeft(node.left)
```

```
def find(n):
    while (n.left != null)
        n = n.left
    return n
```

BST (Binary Search Tree)

for any internal node

- all keys in left subtree are lesser than node's key
- all keys in right subtree are greater than the node's key

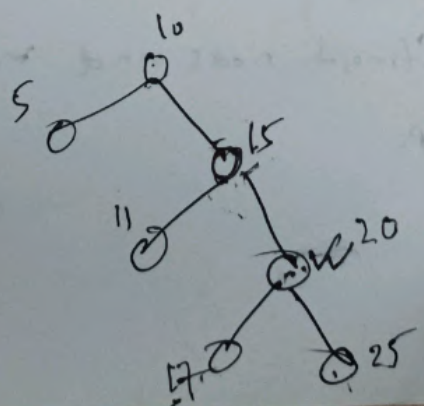
smallest key ⟶ leftmost

largest key ⟶ right most

find successor node (immediatly larger than node)

```
def findSuccessor(node):
    return findLeftmost(node.right)
```

right subtree থাকে তাহলে possible



find a key in BST

```
def findNode(key, node):
    if node == null:
        return null
    elif key == node.element:
        return node
    elif key > node.element:
        return findNode(key, node.right)
    else
        return findNode(key, node.left)
```

How we can get keys in sorted manner

1. in-order traversal
2. Start at leftmost node and visit successor at each step.

29 Aug 2022, 13:39

1. def printsorted(n):

    if n == null:

        return

    else:

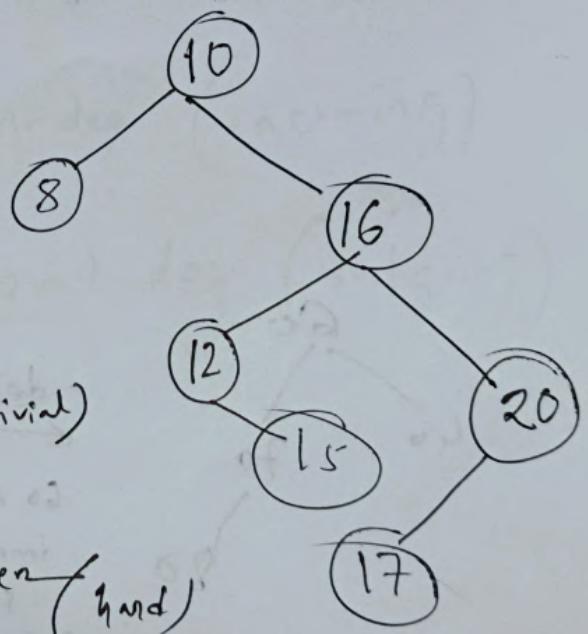
        printsorted(n.left)

        Print(n.element)

        printsorted(n.right)

Insert 17

15

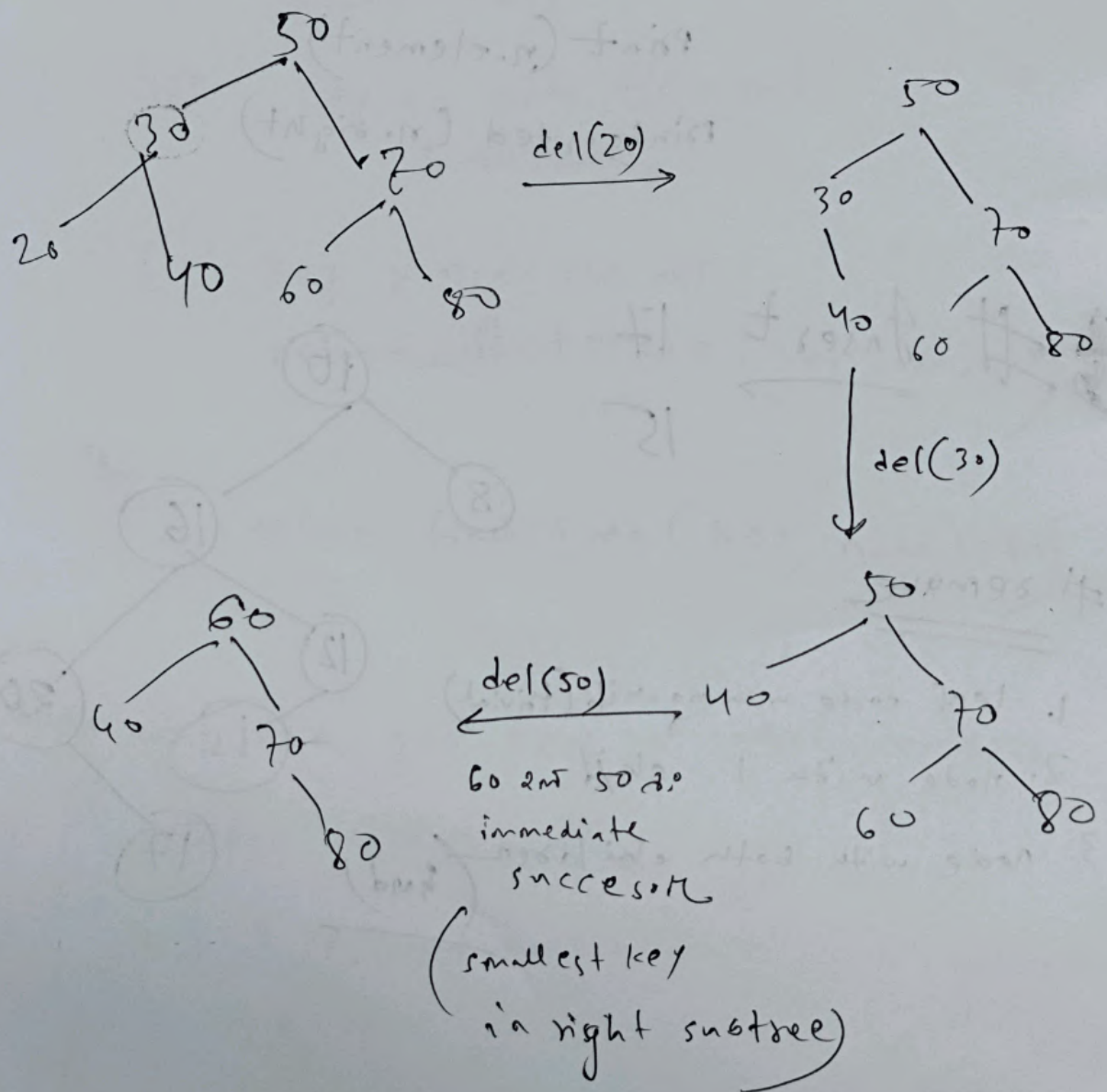


remove

1. Leaf node with no child (trivial)
2. node with 1 child
3. node with both children (hard)

$\log_2 n$ comparison করতে at least BST তে কোনো key searching এ

Deletion/Removal example in BST



29 Aug 2022, 13:39

Balanced BST

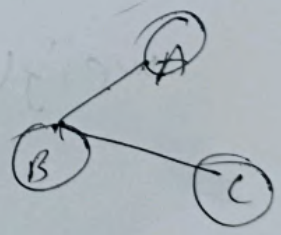
|height of left subtree - height of right subtree| ≤ 1

Graph

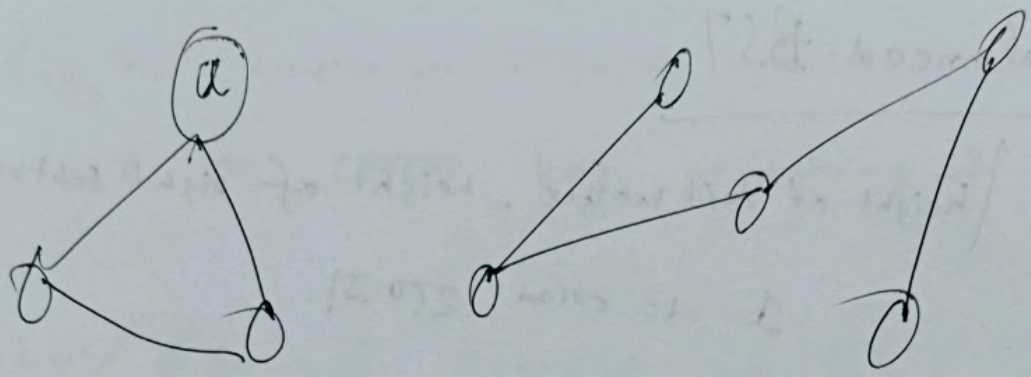
directed
undirected
weighted
unweighted
degree → in-deg (incoming)
→ out-deg (outgoing)

adjacent

$(u, v) \in E$

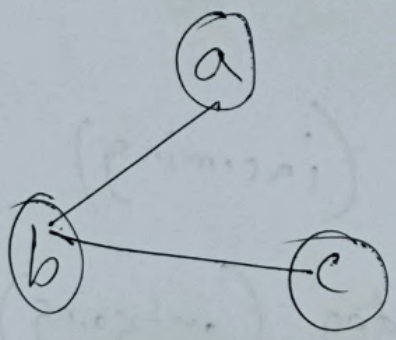


$G = (V, E) = (\{A, B, C\}, \{(A, B), (B, C)\})$



2 connected components of a graph

Adjacency Matrix



	a	b	c
a	0	1	0
b	1	0	1
c	0	1	0

weighted হলে 1 এর জায়গায় weight বসবে।

a ⟶ b

(a, b) → 1

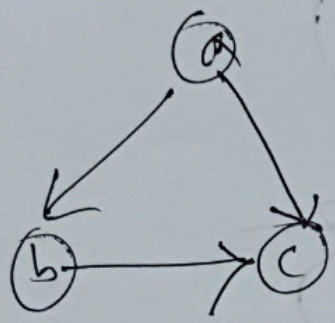
(b, a) → 0

	b
a	1

	a
b	0

29 Aug 2022, 13:39

Adjacency List

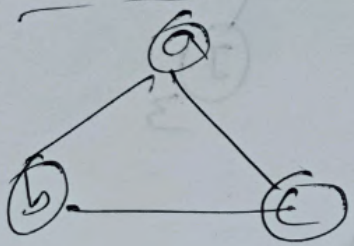


neighbours

a → b → c

b → c

c



a → b → c

b → a → c

c → a → b

matrix better

dense graph $m \approx n^2$

sparse graph $m << n^2$

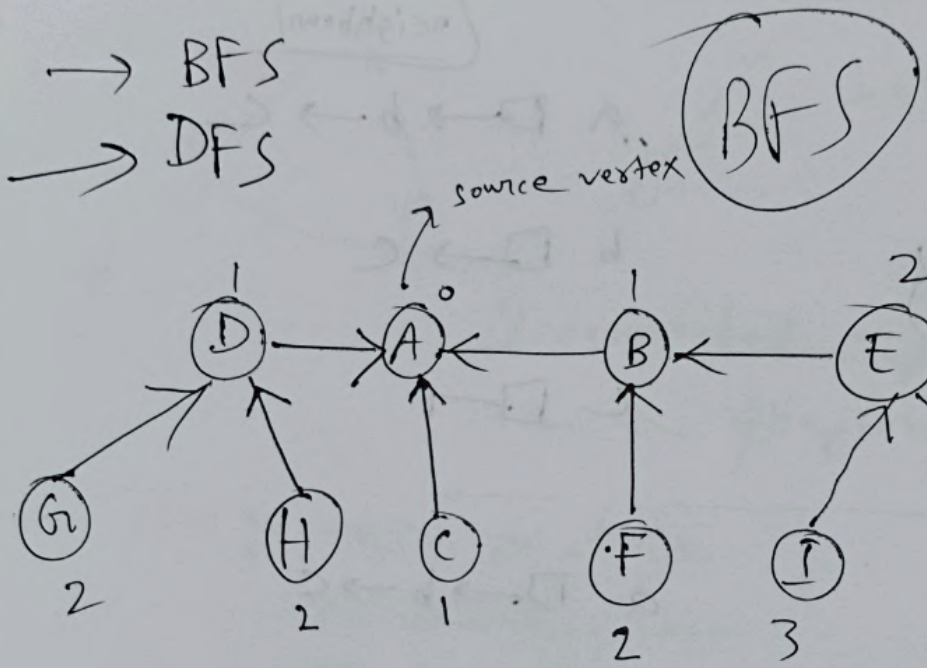
m edge
n vertex

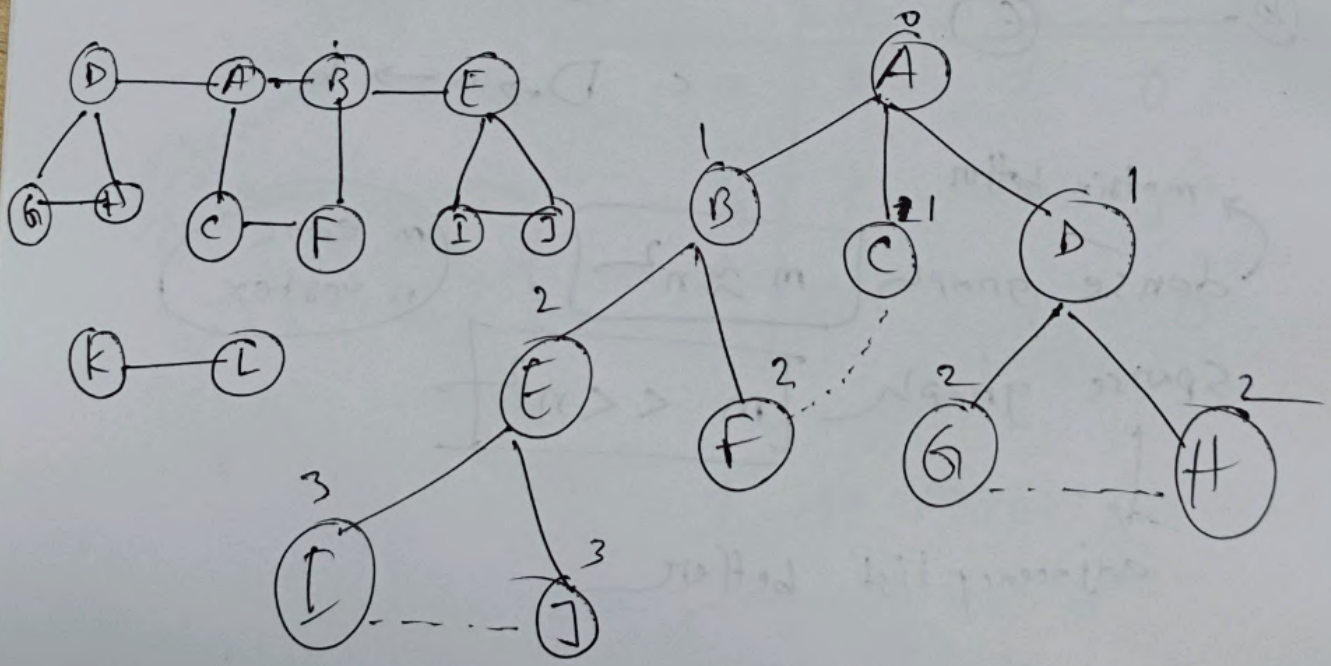
adjacency list better

Graph Traversal

→ BFS

→ DFS



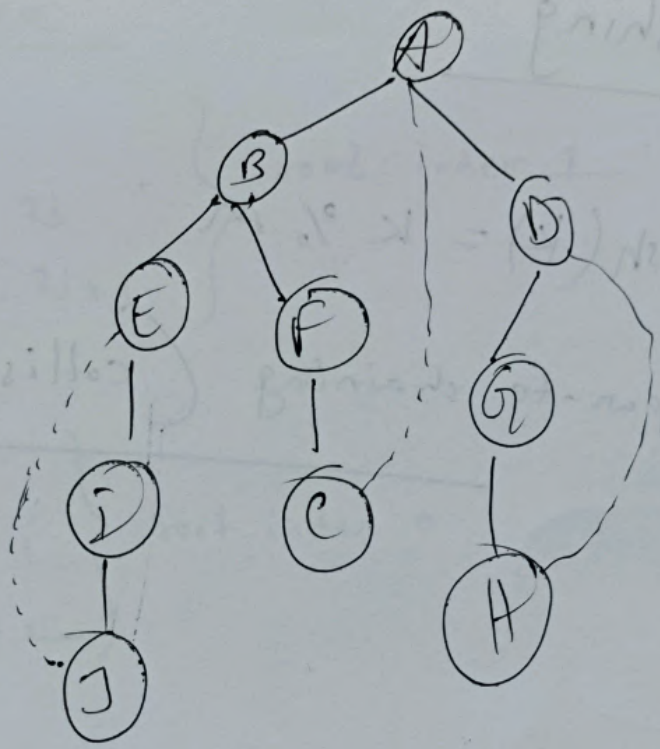


BFS tree

BFS: A B C D E F G H I J

29 Aug 2022, 13:39

DFS



A B E I J F C D G H

discovery / arrival time (✓ time ↓ first time explored)

finish time / departure time (neighbours explored time)

Hashing

hash(k) = k % m

Separate chaining (collision)

29 Aug 2022, 13:39

Draw Binary tree

Parent = i
left child = 2i
right child = 2i+1

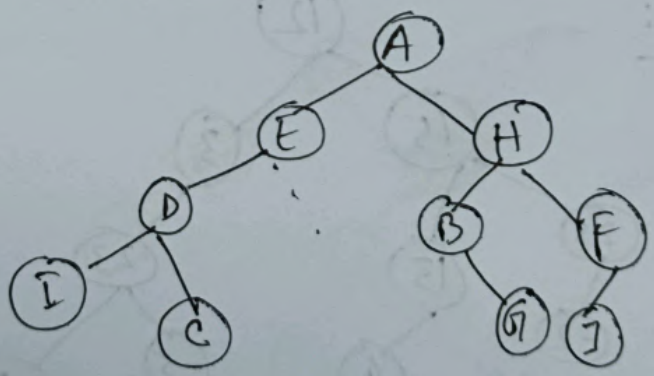
} root index 1

Parent = i
left = 2i+1
right = 2i+2

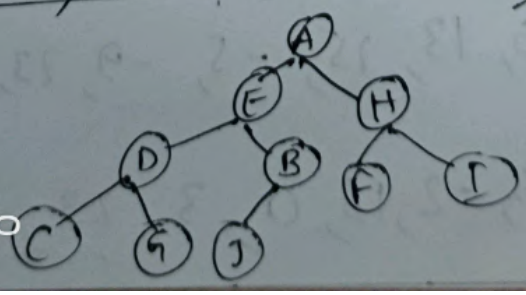
} root index 0

[null, A, E, H, D, null, B, F, I, C, null, null, null, G, J, null]

1) Draw Binary tree



2) complete binary tree (ignore null)



1. adjacency matrix এর

adjacency list করতে হবে

Draw graph

indegree and outdegree of each vertex

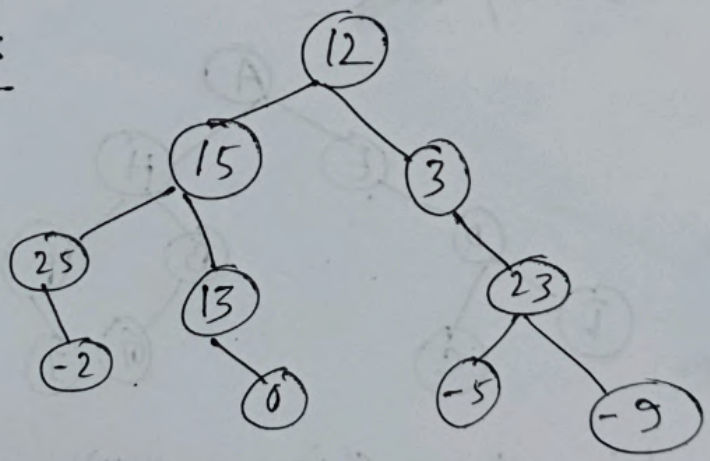
2. Draw BT

Pre-order, in-order, Post-order

Complete BT

3. [null, 12, 15, 3, 25, 13, null, 23, null, -2, null, 0, null, null, -5, -9]

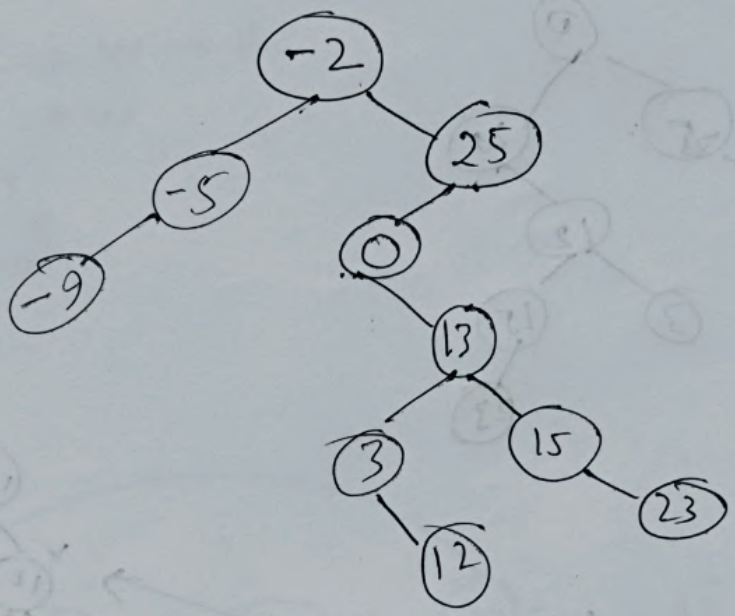
a) Binary Tree:



b) Post order: -2, 25, 0, 13, 15, -5, -9, 23, 3, 12

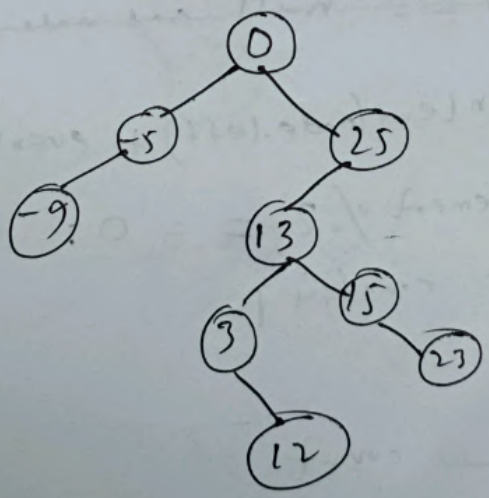
Pre-order: 12, 15, 25, -2, 13, 0, 3, 23, -5, -9

c) using Post order traversals to insert elements in BST.



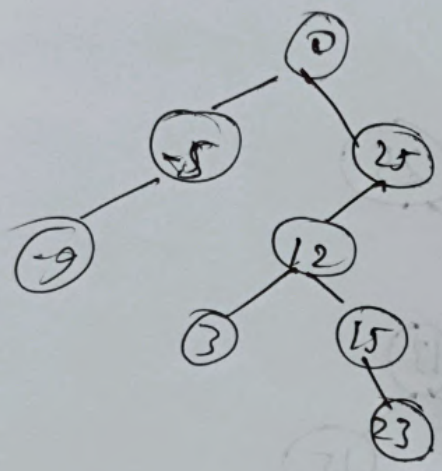
d) deletion in BST

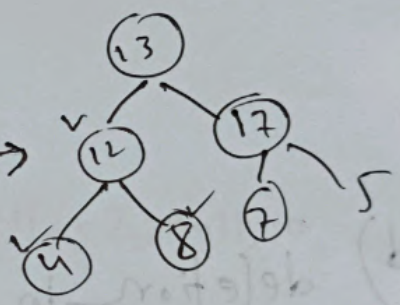
i) Delete (-2) ('succession) → 0



ii) delete 13 (predecessor)

12





AND: 3

```
# def evenLeaf(node):
    if node == null:
        return 0
    if node.left == null and node.right == null:
    count = evenLeaf(node.left) + evenLeaf(node.right)
    if node.element % 2 == 0:
        return count + 1
    else
        return count
```

29 Aug 2022, 13:40

K. → L → S

L → M

S → H

M → S → W → H

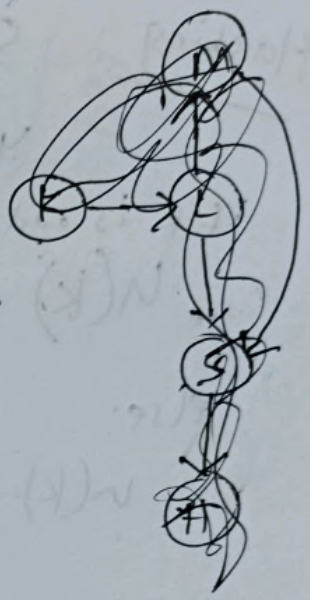
R → M → W

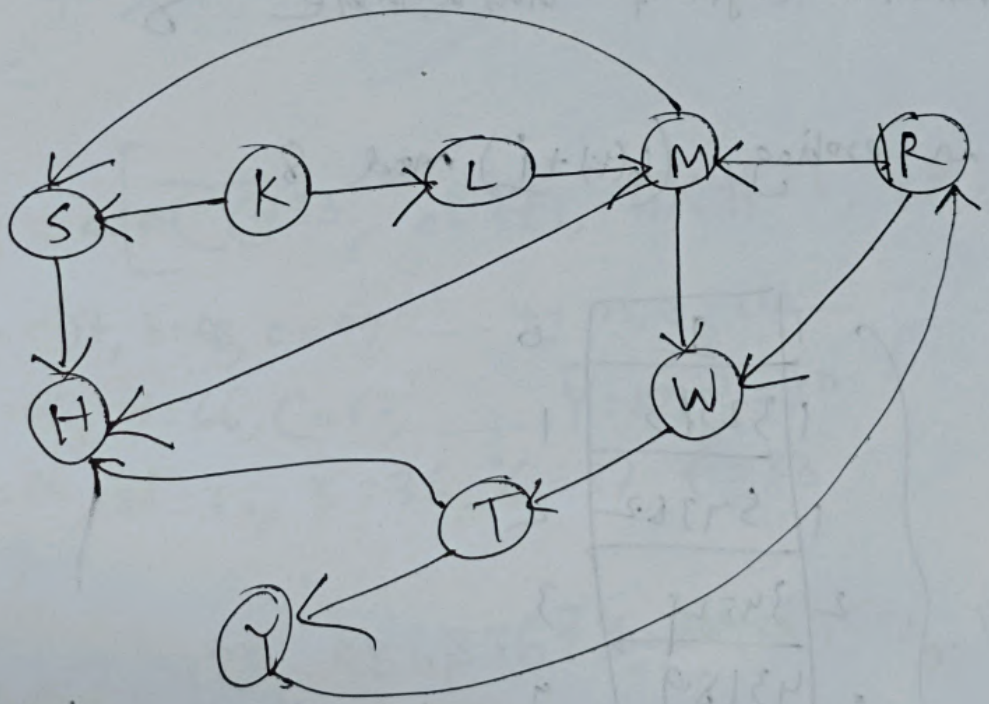
W → T

T → Y → H

Y → R

H





Hashing: 54362, 34521, 3233, 43189, 2275, 1, 672, 8892

if k is odd:

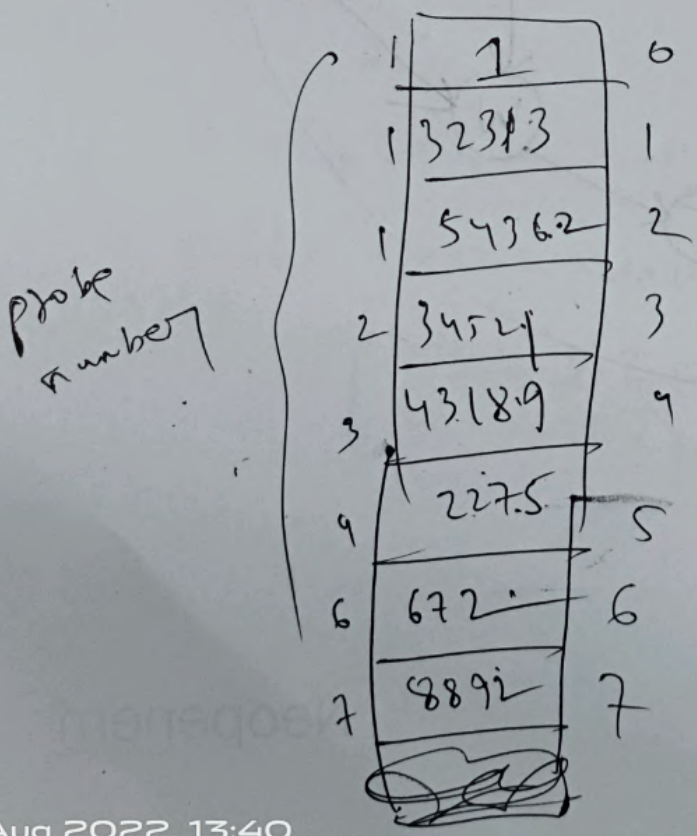
$h(k)$ = # of even digits in k

else:

$h(k)$ = # of odd digits in k

a) # minimum length of hash table 8

b) linear probing $(h(k)+i) \bmod 8$



29 Aug 2022, 13:40

Quadratic probing

$$(h(k) + i^2) \bmod N$$

Linear:

$$(h(k) + i) \bmod N$$

Double hashing:

$$(h_1(k) + i \cdot h_2(k)) \bmod N$$

[a@C@35, bD$74, hX#21, tZ%98, k1€3P]

a = 97, b = 98, c = 99, ... y = 121, z = 122

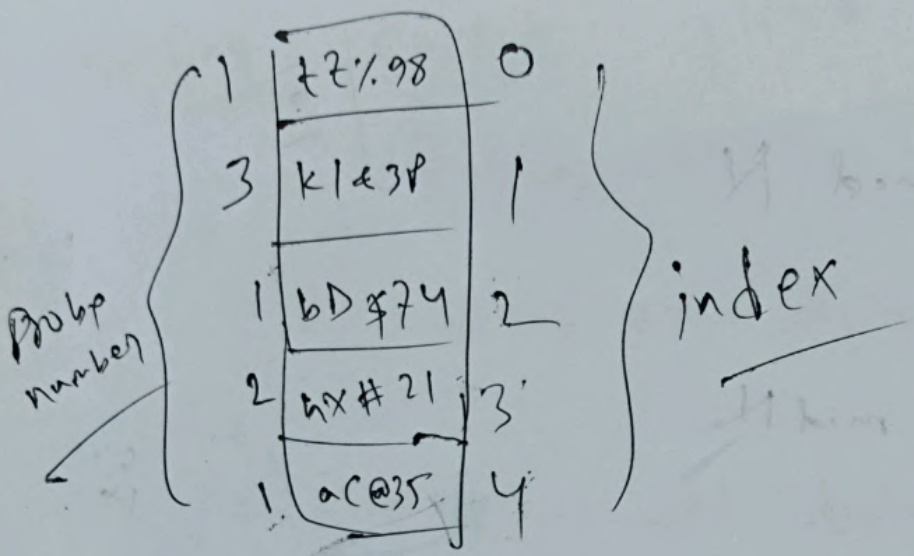
A = 65, B = 66, C = 67, ... Y = 89, Z = 90

@ = 64, # = 35, $ = 36, % = 37, € = 38

0 - 9

[97676435, 98683674, 1048835 21, 116903798, 107138380]

f(word) = | summation of alphabets symbols − Product of digits | % 5



	0	1	2	3	4	5
A =	5,	1,	7,	2,	8,	4

Auxiliary Array

B =	0	1	2	3	4	5	6	7	8
	0,	0,	0,	0,	0,	0,	0,	0,	0
	0,	1,	1,	0,	1	1,	0,	1,	1

Search & Sort

1 2 4 5 7 8

29 Aug 2022, 13:40

AUX

39 40 29 27 27

$f(k) = k \% 5$

0	1	2	3	4
0	29	27	27	39

$(39 - 15) \% 5 = 24 \% 5 = 4$

$(40 - 28) \% 5 = 12 \% 5 = 2$

$(29 - 2) \% 5 = 27 \% 5 = 2$

$(27 - 72) \% 5 = 45 \% 5 = 0$

$(27 - 3) \% 5$

$24 \% 5 = 4$

-60, 30, 55, 55

-12, 6, 11, 11 → 12 + 12 = 24 length

0 —— 23

$-12 + 12 = 0 \rightarrow 1$

$6 + 12 = 18 \rightarrow 1$

$11 + 12 = 23 \rightarrow 2$

Aux array