CH 1 INTRODUCTION

- ✓ TYPES OF DATA STRUCTURES
- ✓ ALGORITHM ANALYSIS
- ✓ ASYMPTOTIC NOTATIONS

ROW ORDER , COLUMN ORDER

- ✓ SPARSE MATRICE & REPRESENTATION

DATA STRUCTURE - way of organizing all data with their relationships to each other

TIME COMPLEXITY - It quantifies amt of time taken
SPACE COMPLEXITY - It quantifies amt of space/memory taken by algorithm

RUNTIME - Amt. of time taken by program to execute with given input to get the output

BIG OH
$f(n)$ is $O(g(n))$: $f(n) <= g(n)$
BIG OMEGA
$f(n)$ is $\Omega(g(n))$: $f(n) >= g(n)$
BIG THETA
$f(n)$ is $\Theta(g(n))$: $f(n) = g(n)$

Eg. $2n+10$ is $O(n)$

$2n+10 \leq c(n)$

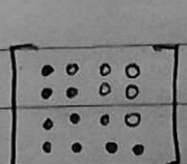
$c(n) - 2n \geq 10$

$n(c-2) \geq 10$

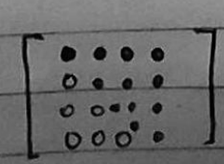
$n \geq 10/(c-2)$

PICK $c=3$ $\therefore$ $n_0 = 10$

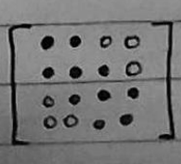
SPARSE MATRIX - matrix that has large no. of zeroes



LOWER Δ



UPPER Δ



TRI-DIAGONAL

NO. OF ROWS (M) / COLUMS (N) = Upper bound - lower bound + 1

FOR 2D :

ROW MAJOR = $BA + \omega (N(i-1) + (j-1))$

COLUMN MAJOR = $BA + \omega ((i-1) + M(j-1))$

FOR 3D :

ROW MAJOR = $BA + \omega [N(i-LBR) + (j-LBC) + MN(K-LBD)]$

COLUMN MAJOR = $BA + \omega [(i-LBR) + M(j-LBC) + MN(K-LBD)]$

size of element

ROW COLUMN DEPTH

CH 2.1 STACK

- ✓ PUSH - POP - PEEP - CHANGE
- ✓ APPLICATIONS
- ✓ REURSIVE FUNCTION TRACING
- ✓ PRINCIPLES OF RECURSION
- ✓ REMOVAL OF REURSION
- ✓ TOWER OF HANOI
- ✓ INFIX, POSTFIX, PREFIX & CONVERSTION

STACK → follows LIFO (where insertion & del. takes place from 1 end)
Operations : PUSH - POP - PEEP

OVERFLOW = ~~STAR~~ stack pointer exceeds fixed memory
UNDERFLOW = when empty stack is popped

RECURSIVE FUNCTION TYPES:

DIRECT - if function calls itself
INDIRECT - if fun1 calls fun2, which ultimately calls fun 1.
TAILREC. - if no operation are pending when recursive func. returns to its caller. Eg.

```
int fact1 (int n, res)
{ if (n==1)
    return res;
  else
    fact1(n-1, res*n);
}
```

EXCESSIVE - multiple call to itself in function body
LINEAR - when function only calls only for 1 time or pending operation does not make recursive call
TREE RECURSIVE - if pending operation makes another recursive call to function.

REMOVAL OF RECURSION USING - ITERRATION
- STACK

TOWER OF HANOI - only 1 disk (uppermost) at a time
- large disk cannot be set on smaller

```
Tower (n, A, C, B) {
  if (n == 0)
    { print "S to d";
      return; }

    tower (n-1, A, B, C);
    print "S to d";
    tower (n-1, B, C, A);
}
```

PREFIX EVALUATION

+9*26 ←

OPERATION	STACK
6	6
2	6 2
*	12
9	12 9
+	21

POSTFIX EVALUATION

10 2 8 * + 3 - →

OP	STACK
10	10
2	10 2
8	10 2 8
*	10 16
+	26
3	26 3
-	23

INFIX TO POST FIX

op stack expression

sign	priority
^	3
*, /	2
+, -	1

INFIX TO PREFIX

→ reverse
→ convert to postfix
→ reverse

CH2.2 QUEUE

- ✓ INSERT & DELETE
- ✓ CIRCULAR ~~QUE~~ QUEUE
- ✓ PRIORITY QUE
- ✓ DOUBLE - ENDED
- ✓ APPLICATIONS

QUEUE - linear DS - Insertion from rear end
- follows FIFO - deletion from front end

Total elements in queue = rear - front + 1

ENQUE = inserting element DEQUE = deleting element

Drawback of Simple Queue → after deletion space is waste

TYPES OF QUEUE

1. CIRCULAR – linear DS, FIFO, last pos. connected to 1st pos
   - also called Ring buffer
   - utilizes space fully (advantage)

2. PRIORITY – ds having collection of elements which are associated with priority
   Eg. scheduling job in os
   - use to manage limited bandwidth for transmission
   - ASC. PRIORITY → only smallest element can be remov
   - DESC PRIORITY → only largest element can be remove

3 DOUBLE-ENDED – insertion & del. from both ends

- INPUT RESTRICTED → only input at one end
- OUTPUT RESTRICTED → only deletion from one end

CH 2.3 LINKED LIST

- ✓ MEMORY REPRESENTATION
- ✓ SINGLY
- ✓ DOUBLY
- ✓ CIRCULAR
- ✓ APPLICATIONS

ADV.
- dynamic size
- easy ins/del

DISADV.
- extra space for *
- random access ✗
  ∴ no binary search

LL - Linear DS, elements, stored ~~and~~ ~~not~~ are not in contigous memory location
- elements linked using pointers
- first node = HEAD

Each node has 2 parts DATA | LINK

TYPES OF LINKED LIST

1. SINGLY - each element points to next element
2. DOUBLY - points to next & prev.
   - can travel in both dir
   - inserting node by another is easy
   - delete is easy if pointer is given
   - (dis adv) req. extra space

3. CIRCULAR LINKED-LIST - nodes connected to form O
   - no null at end
   - can be singly / doubly

MEMORY PRESENTATION

	INFO	LINK
1001	A	1002
1002	D	1003
1003	B	NULL

1002

CH 2 APPLICATIONS

STACK
- reverse string
- infix to postfix / prefix conversion
- tower of hanoi
- to undo any mechanism
- pile of dinner plates

QUEUE
- printing
- task scheduling
- call center phone sys.
- used in ~~call center~~ networking to handle traffic

LINKED LIST
- dynamic memory allocation
- maintaining directory of names
- to implement stack & queue
- Image viewer
- Music player

CIRCULAR LINKED LIST
- to share time b/w users by os
- multiplayer games
- snake game on phone
- doubly O is used to implement fibonacci heap.

CH 3 APPLICATIONS

TREE - storing hierarchical data (HTML, FOLDERS)
- used in indexing database
- syntax trees in compilers
- used for quick searching

HEAP - in algo (Dijkstra's)
- to implement priority queue
- heap sort

GRAPH - google maps
- represent flow of computation
- GPS navigation
- path optimization algo

CH 5 HASHING

→ quick searching & retriving info
→ database indexing
→ symbol tables
→ file & directory
→ store massive info

CH3.1 ~~GRAPH~~ TREE

- ✓ TRAVERSEL
- ✓ BST
  THREADED BINARY TREE
- ✓ AVL TREE
- ✓ APPLICATIONS

HEIGHT = no. of levels
NODE DEGREE = no. of children
TREE DEGREE = max. node degree

B-TREE → tree with atmost 2 children
→ It can be empty
→ subtree of B-tree are also orderes

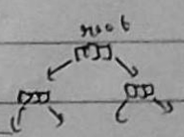
] diff with TREE

MIN nodes at height h = h
MAX nodes at height h = $2^h - 1$ (complete B-tree)

parent of left child i = i/2 (unles i = 1)
right child = i+1

if parent = i ∴ LEFT = 2i , RIGHT = 2i+1

B-TREE ⟶ array representation
→ Linked representation



PREORDER : A B C
INORDER : B A C
POSTORDER : B C A

ANCESTOR of (Z) → X, Y ↓
DESCENDENTS of (X) → Y, Z

MAX. no. of nodes on level i = 2^{i-1}

let n_0 = leaf node, n_2 = 2deg. node ∴ $n_0 = n_2 + 1$

BINARY SEARCH TREE
- unique key
- left subtree < right subtree
- LEFT & RIGHT are subtree

INSERTING → on bases of searching ; put on L/R

DELETING → LEAF NODE
↳ simply delete
→ 1 CHILD
↳ delete & join child
→ 2 CHILD
↳ delete & replace with highest (LST)
" " lowest (RST)

HEIGHT OF BST with n elements can be n

SEARCHING, INSERTION, REMOVAL
→ O(h), h is height
If max/min element is deleted, than $O(\log_2 n)$
worst $O(N^2)$

AVL TREE
- height of leaf is 1
- O(log N) for search, insert, delete
- worst case O(N) as height can be N-1

BAL. FACTOR = h(LST) - h(RST)

TYPES OF ROTATION
LL, RR, LR, RL
sigle rotation O(1) tim

Eg. Insertion & deletion from PPT

CH 3.2 HEAP

- ✓ PRIORITY QUEUE
- ✓ BINARY HEAPS
- ✓ TREAPS

MAX HEAP - parent node > child nodes
MIN HEAP - parent node < child node

Heaps are implement for priority queues (only way)
↳ ~~INSERT~~ DELETE with high/low priority
↳ INSERT with any priority

~~DELETION~~

TREAP - ds which combines search BST TREE & BINAR HEAP
↳ inserting as BST & then rotating for setting priority

CH 3.3 GRAPH

- ✓ MEMORY REPRESENTATION
- ✓ BFS & DFS
- ✓ SPANNING TREES (all 3 algo)
- ✓ shortest path tree
- ✓ APPLICATIONS

GRAPH - consist vertices & nodes — UNDIRECTED
- subset of tree — DIRECTED
- no uniaue node like root

GRAPH REPRESENTATION → adjacency matrix (sequential)
→ adjacency list (linked list)

MATRIX		LIST
A—B	M[A][B] = 1 M[B][A] = 1	
A→B	M[A][B] = 1 M[B][A] = 0	

- BREADTH FIRST SEARCH → write all connections 1st
- DEPTH FIRST SEARCH → explore the connected elements

→ spanning tree for both

SPANNING TREE - It is subset of graph
- which covers all vertices but should not have cycles

for undirected = n^{n-2}

If it has N nodes, edges N-1

FOR MIN. SPANNING TREE

1. KRUSHKALS - remove parallel edges & loop
- arrange in asc (weight)
- start taking small edges but should not make cycle.

2. PRIMS - remove all parallel edges & loop
- take root node & start taking small edg
- always choose min & include all vertice

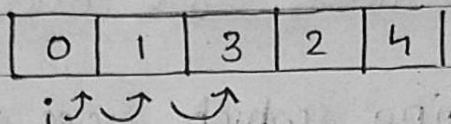
3. DIJKSTRA - make table.
- open point starting from a point
- mark shortest dist to all points.

CH 5 SEARCHING

- ✓ LINEAR SEARCH
- ✓ BINARY SEARCH

LINEAR SEARCH - process of sequentially checking elements of list until finding the match.

FIND 3 in



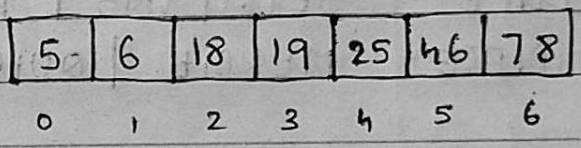
O(n)

BINARY SEARCH - It finds position of a target value in sorted array only.

O(logn)

→ It is FASTER than Linear search

Eg. SEARCH 6 in



LEFT = 0 RIGHT = 6,

MIDDLE INDEX = 6/2 = 3 ∴ MIDDLE = 19

∵ 6 < 19

LEFT = 0 RIGHT = MIDDLE INDEX - 1 = 2

MIDDLE INDEX = 2+0/2 = 1

∴ MIDDLE = 6 (KEY)

∵ ELEMENT FOUND