Short Notes & Formula Revision: Data Structure

Special class

Vishvadeep Gothi • Jan 26, 2022

DS:- 2 PM
OS :- 3:15 PM
COA :- 4:45 PM

DS: Short Notes

By: Vishvadeep Gothi

1. Thank you
2. ____________

unacademy

VISHVADEEP GOTHI

* Computer science Head, Unacademy Brands
* Top Educator (#3 for CS) on unacademy plus for GATE/ESE
* ME IISc Bangalore
* MTech Bits Pilani (Data Science)
* GATE AIR-19 (in 4th year), 682 (in 3rd year), 119 440
* 9 Years of GATE/ESE teaching experience (Gateforum, THE GateAcademy, ACE Academy)
* 13 Years of teaching experience
* 1 year Industry experience for Software Development

Vishvadeep Gothi

Array

- $Loc(A[i]) = Base + w * (i - LB)$
- 2-D array RMO: $Loc(A[i][j]) = Base + w * [(i - LBi) * n + (j - LBj)]$
- 2-D array CMO: $Loc(A[i][j]) = Base + w * [(j - LBj) * m + (i - LBi)]$

$m \times n$

Operation	Comparisons	Space	R.T.C.
Find Minimum	$N - 1$	1	$\theta(n)$
Find Maximum	$N - 1$	1	$\theta(n)$
Find Min Max Both (N is even)	$1.5N - 2$	N	$\theta(n)$
Find Min Max Both (N is odd)	$\lceil 1.5N \rceil - 2$	N	$\theta(n)$
Find Second Minimum	$N + \log n - 2$	N	$\theta(n)$
Find Second Maximum	$N + \log n - 2$	N	$\theta(n)$

- Linear Search: O(n)
- Binary Search: O(log n) → array should be sorted

Find min with Tournament $N - 1$ N $\theta(n)$

Vishvadeep Gothi

Linked List

Insertion (Singly)	Complexity
1. At beginning	Constant
2. After a given node	Constant
3. Before a given node	O(n)
4. At the end	Theta(n)
5. At the end (last node add. given)	Constant
6. In a sorted list	O(n)

Insertion in Circular List (Singly)	Complexity
1. At beginning	O(n)
2. After a given node	Constant
3. Before a given node	O(n)
4. At the end	Theta(n)
5. At the end (last node add. given)	Constant
6. In a sorted list	O(n)

Vishvadeep Gothi

Linked List

Deletion (Singly)	Complexity
1. At beginning	Constant
2. After a given node	Constant
3. Before a given node	O(n)
4. At the end	Theta(n)
5. At the end (last node add. given)	Theta(n)
6. In a sorted list	O(n)
7. Of a given node	O(n)

Vishvadeep Gothi

Linked List

Insertion in Doubly List	Complexity
1. At beginning	Constant
2. After a given node	Constant
3. Before a given node	Constant
4. At the end	Theta(n)
5. At the end (last node add. given)	Constant
6. In a sorted list	~~Constant~~ O(n)

Deletion in Doubly List	Complexity
1. At beginning	Constant
2. After a given node	Constant
3. Before a given node	Constant
4. At the end	Theta(n)
5. At the end (last node add. given)	Constant
6. Of a given node	Constant

circular doubly linked list $\Rightarrow$ $\Theta(1)$ for all operatns except insertn in sorted list.

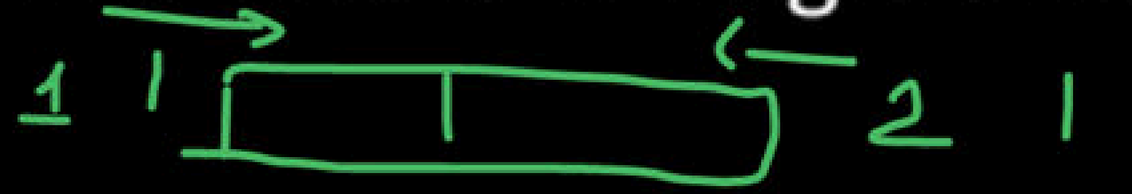
Vishvadeep Gothi

Queue & Stack

- Queue using array: Enqueue() & Dequeue in constant time
- Queue using linked list with 2 pointers(first & last)
  - Enqueue() & Dequeue in constant time with insertion a last and deletion from front

Double Ended Queue	Enqueue	Dequeue
Input Restricted	Rear	Both
Output Restricted	Both	Front

- Stack Using Array: PUSH & POP in constant time
- Stack using linked list: PUSH & POP in constant time (with both from starting)
- Valid stack permutations = $\frac{{}^{2n}C_{n}}{n+1}$ $= \frac{2n!}{n! * n! * (n+1)}$
- Invalid stack permutations = $n! - \frac{{}^{2n}C_{n}}{n+1}$
- Two stacks in single array: Overflow condition: $top1 = top2 - 1$ or $top2 = top1 + 1$



Vishvadeep Gothi

Tree

n_1

n_2

- N = L + I
- Binary tree
  - $I = I_1 + I_2$
  - $L = I_2 + 1$
  - $N = 2*I_2 + I_1 + 1$
  - n3 = n1 – 2 n3 = nodes with degree 3, n1 = nodes with degree 1 (degree = neighbours
  - Max nodes with height h = $2^{h+1} - 1$ H (T) = 0 with 1 node
  - Min nodes with height h = h + 1
  - Max nodes at level L = 2^L
  - Min nodes at level L = 1
  - Number of BT with n unlabeled nodes = $\frac{2nC_n}{n+1}$
  - Number of BT with n distinct keys = $\frac{2nC_n}{n+1} * n!$
- First Symbol of preorder is root
- Last Symbol of postorder is root

Vishvadeep Gothi

Tree

- Reverse of Converse Preorder is conventional Postorder
- Reverse of Converse Postorder is conventional Preorder
- Reverse of Converse Inorder is conventional Inorder
- Preorder and Postorder can provide unique tree if all internal nodes have maximum allowed children; otherwise inorder is required to get unique tree along with either preorder or postorder.
- CBT: Max nodes with height h = $2^{h+1} - 1$
- CBT: Min nodes with height h = 2^h
- Array representation of tree:
  - Root at index 1
  - Left child of node at index i = 2i
  - Left child of node at index i = 2i+1
  - Parent of node at index i = $\lfloor i/2 \rfloor$

Tree

- Full BT (2-Tree)
  - L = I+1
  - N = 2I + 1
  - N = 2L – 1
  - N is always odd
- K- tree (every internal nodes has k children)
  - L = (k-1)I + 1
- BST:
  - Inorder is sorted sequence in ascending order
  - If preorder given then insert keys from first to last and construct tree
  - If postorder given then insert keys from last to first and construct tree
  - Number of BST with n distinct keys = $\frac{^{2n}C_{n}}{n+1}$

Vishvadeep Gothi

Tree

balanced BST

AVL Tree

$O(\log n)$ $h \simeq \log n$

$O(\log n)$

- BST:

Case	Searching, Insertion & Deletion
	O(h)
Average	O(log n)
Worst	O(n)

- AVL Deletion:

Case	Rotation
R0	LL
R1	LL
R-1	LR
L0	RR
L1	RL
L-1	RR

$$n_{min}(h) = \begin{cases} 1 & h = 0 \\ 2 & h = 1 \\ n_{min}(h-1) + n_{min}(h-2) + 1 & h > 1 \end{cases}$$

- Min height of an AVL tree with n nodes = $\lfloor \log_2 n \rfloor$
- Max height of an AVL tree with n nodes $\approx 1.44 \log_2 n$

Vishvadeep Gothi

Graph & Hashing

Traversal	Adjacency Matrix	Adjacency List
BFS	$O(\|V\|^2)$	$O(\|V\|+\|E\|)$
DFS	$O(\|V\|^2)$	$O(\|V\|+\|E\|)$

- **Disadvantages of open addressing:** (closed hashing)
  1. Collided records require more probes
  2. Deletion not possible
  3. Overflow problem
- **Advantages of chaining:**
  1. Collided records require less probes
  2. Deletion possible
  3. No overflow problem
- Load Factor = $\frac{Total\ Keys}{Total\ Slots}$
- Space Utilization= $\frac{Occupied\ Slots}{Total\ Slots}$

$0 \leq L.F.$ chaining

$0 \leq L.F. \leq 1$ for open addressing

$0 \leq S.U. \leq 1$

Vishvadeep Gothi

Graph & Hashing

Advantages: chaining

1. Simple to implement.
2. Hash table never fills up, we can always add more elements to the chain.
3. Less sensitive to the hash function or load factors.
4. It is mostly used when it is unknown how many and how frequently keys may be inserted or deleted.

Disadvantages: chaining

1. Cache performance of chaining is not good as keys are stored using a linked list. Open addressing provides better cache performance as everything is stored in the same table.
2. Wastage of Space (Some Parts of hash table are never used)
3. If the chain becomes long, then search time can become O(n) in the worst case.
4. Uses extra space for links.

3:15PM OS

4:45PM → COA

Happy Learning.!