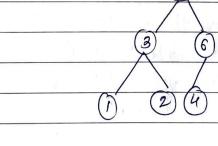
CHAPTER - 7 PRIORITY QUEUES AND HEAPS

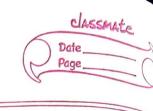


A priority queue is a datastructure that supports the operations Insert and Deletethin (return 4 remove minimum elemen or Deleternax (return & remove maximum element) # Main Pa Operations Insert (key, data): Element are ordered based on key. Deletethin/ Deletethax! Remove 4 return minimum/maximum element Gettnin/Gettnax: Return minimum/maximum element. # Auxillary pa Operations kth largest/kth smallest Size: Returns the number of elements in Pa Heap Sort: Sorts pa element based on (key) Heaps 4 Binary Heaps Heap! A tree with below properties - Value of a node must be(>,) or (<) to its children. - It should form a complete binary tree.



Heap Types: Min heap! Value of node (5) to its children
Than heap! II -1- 10 (7) 11- 11-

-H-	A two
#	Binary Heaps (Each node may have upto children)
	1 40 (13) In a grant of the same
	A londer of the second
	13 6 Le maples
	(2)1-3) (2)
	17/6/ 17/13/6/1/4/2/5
	17/6 17/13/6 1 4/2/5 0 1 2 3 4 5 6
#	Representation: Note that and the state of t
	type struct Heap ?
	amay [] int
	count int I/No. of elements in Heap
	capacity int 11 Size of the heap
	heaptype int Il Heap true minima
	heaptype int // Heap type min/max
	1 = [2] = 1
#	Creating a Heap
	func Create (canacity hoursens int) + Hear ?
	Creating a Heap Func Create (capacity, heapsype int) * Heap? h:= 4 Heap?? h. array = make ([] int, capacity)
	h. array = make (r) int, capacity)
	h. count = 0
	h. heaptype = heaptype 11 Min = 0, than = 1
3	return h
	2
-	The Sightchild (h "Hanp i red) not?
#	
1,	Parent og a Node: It is at location e-1 in the away. For example => 13 is at location 1 i.e. e=1
	: Parent of $13 = \frac{\dot{e}-1}{2} = \frac{1-1}{2} = \frac{7}{2}$
	=> array [0] => 17



Tale cannot find the parcut for e=0 4 e>= h.count thur func Parent (h *Heap, è int) int ?

if ex=0 || e>= h.count? return (e-1/2) # Children as a Mude For a node at ith position, it children are at 2xit 2 + è+2 locations. 13 is at position => 1 del your It children are at which do your Jones of the heart 2 + e + 1 => 2 × 1 + 1 => 3 (Left child) 2 + é + 2 = 2 + 1 + 2 =) 4 (Right child) h.amay [3] = 1 h. amay [4] = 4 renting a map func Leftchild (h * Heap, è int) ents En left: = 2 +(estalas torra som = porno if left 7= h. count ? return -1 white and - havely be 11 Min = a thora func Rightchild (h + Heap, e eut) eut ? right := 2 + e +2 if right >= h.count ?

return right

4



#	Maximum Element: In max heap, the max element is always
	at the root.
	Light interpretation of the control
	func Maximum (h + Heap, i) int & if h.count == 0 {
	if h.count == 0 f
	return -1
	4
	return h. array [0]
	3 307
#	Heapifying an Element
++	an clement
	1) To heapify an element, gind the maximum ag it
	children & swap with the element.
	2 Continue the above process until, the element satisfier the
	heap properties.
	Maria Com Transfer Com
4	Algorithm Could ith assistant from water has a faire phould
	1) Accept heap 4 ith position from where heapifying should
	Of the letter of the dillet on in
	(2) Get the left 4 right child as is' (3) If left child exists 4 it is greater than the value at i
	13) IF LEFT CHILD EXIST 4 IT IS GREATER UNAN THE VAILE OF E
	max = left child to product the
	else max = phonon of the state
7 .	(2) IC and the second of the s
	(4) If right child exists 4 it is greater than the value at
	max = Y side could format &
	5) If more to is not equal to e, swap element at e
	with element at mare.
	6 Cau Percolate Down (h, max)
	(2) DONE

	func Percolate Down (h * Heap, è éut) ?
	l teft child := Left child (h, e)
	~ · - Right Child (l. i)
	w mare: = 0
	11 Han to Blackform Sten 3
	if e1=-1 4f h.away[e] > h.away[e] ?
	17107C = X
	3 else {
	mazzė
	3
	11 Algorithm Step 4
	Il Algorithm Step 4 ig r = 1 4f h. away [r] > h. away [max] ? max = r
	max = x hanges and allow some of morbids
	is contine the observe and the depent sol
	11 Algo. Step 5
	ij max != è ?
	h.amay [i], h.amay [mane] = h.amay [mane], h.amay [i]
	I great heart it position and where the proof of and
	Percolate Down (h, max)
J	Time Complexity: O(logn)
	Time Complexity: O(logn)
	The Maril
	Deleting our Element 10/10/2 x 2000
	Algorithm:
	Delete the first (root) element i.e. replace it with
1	heap're last element.
	2) Delete last element
	3 Change court variable
7	4) Heapily the element => Percolate Ducan (h, 0)
	(4) Heapity the element -> PercolateDown (h, 0)

m Find affer and to war

#

	func Delete (h * Heap) int ?	
j h.co	ut==o max := h.amay [o]	
return	- : h.array to] := h.array [count-] Time Complexity:	
	h.amay = h.amay [:count-1] 0 (log n)	
	h. count -= 1	
	Percolate Down (h, o)	
	return max	
	3	
#	Inserting an Element	195
	Algorithm:	
	(1) Insert element at last endex.	
	2) 8 tart heapifying from last index	
	2) Start heapifying from last index. (3) Increase heap capacity before insert.	
	func Resize (h + heap) {	
	old Array := h. array	
	new Capacity := 2 + h. capacity	_
and the second second	new Capacity := 2 + h. capacity h. array := make (E) int, new Capacity)	_
	jer i, v := range oldArray? h.away[i] = oldArray V	(
	n. away (1) = otel Among V	
	1	
		_
	func Track (la + hann 1 1 " 1) }	_
	func Insert (h * heap, data int) { if h.count == h.capacity {	
	Resize (h)	
	3	-
	h.count++	
	i = h. count -1	

11 Percolate Up as de (goally d) a b. array [i] = h. array [e-1/2] {

h. array [i] = h. array [e-1/2]

l = e-1/2 h.away [i] = data I be when bun hast index Em EN : Yange other in ? her beech (h Heap data inch