6-3

n collected block type #

Quiz 1:

Problem 1: (a) Zhang Wenhao

gumi.zwh10@gmail.com

(b)

Problem 2:

(i) Worst-case: & Expected: O(n)

k = max(X) O(n) {}

H = { } ⇒ : O(n)

worst-case: O(n)

⇒ $O(n^2)$

(b) ~~DEF~~

i) Worst-Case:

A. append(o) : amortized time $O(k)$

$(O(n) + O(k) = O(n+k)$

ii) expected: $O(n+k)$

Problem 3. Haphazard Heap (3 parts)

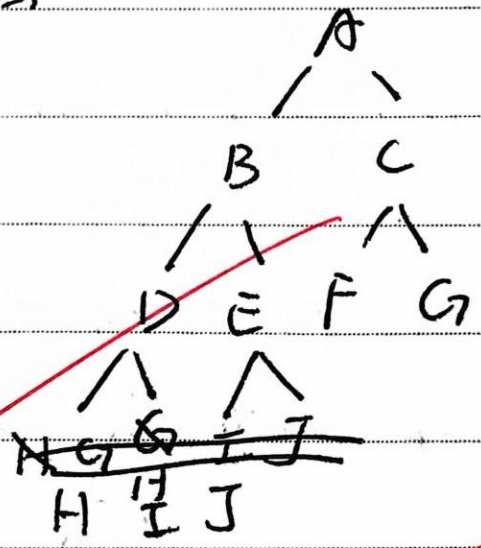
Binary min-Heap. 10 items.

(a) the smallest integer

A

(b) the third smallest integer

B OR C, OR D, E, F, G



(c) the largest integer

~~I OR J OR G OR H. OR F OR G~~

F, G, H, I, J, all leaves can be largest

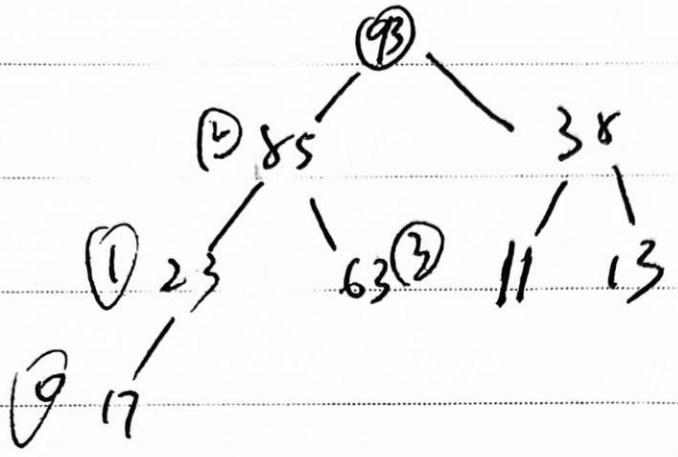
∋ buttom level

Mo | Tu | We | Th | Fr | Sa | Su

Memo No. _____

Date / /

Problems 4 Transforming Trees.



a binary-max Heap

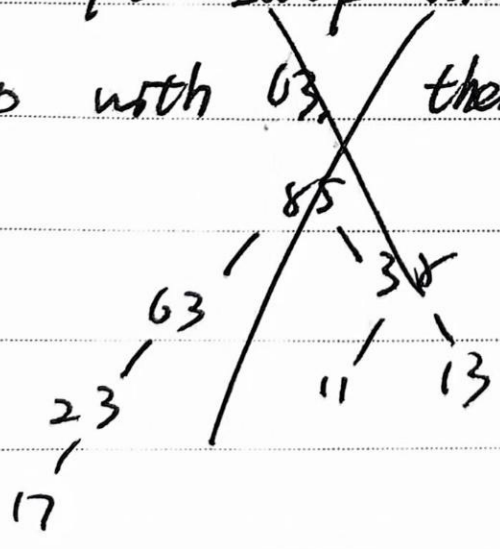
(a)

the array:

93, 85, 38, 23, 63, 11, 13, 17

operat A.delete_max (1):

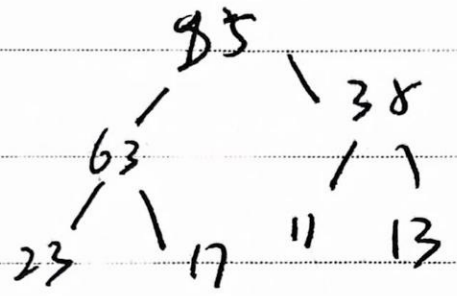
93 swap with 93.left, = then swap with 63, then delete



Mo | Tu

Date / /

H. delet_max() ① swap 93 and 17 first then process max_heapify_down in 17

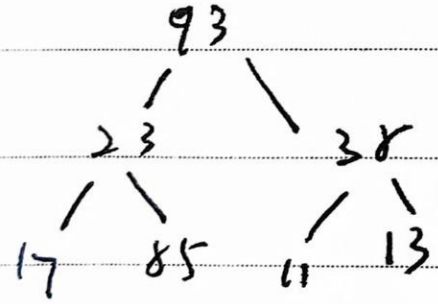


so the array: 85, 63, 38, 23, 17, 11, 13.

b) Suppose it's a sequence AVL Tree S

S. delete_at(3) ⇒ means delete 63.

~~⇒ roterte~~ right roterte 85



⇒ 17, 23, 85, 93, 11, 38, 13

the leaves in traversal order:

or 17, 85, 11, 13

Date / /

Problem 6. Sorting Sock

n students sorted into four houses

(a)

friend number can be determined in O(1) time because friend number of every student is a number in $[0, n-1]$, so we can use counting sort which take $O(n)$ time.

(b) By their books

from question we can know compare operation is in O(1). So we can use merge sort with $O(n \lg n)$ time

(c) bravest

because student's ~~bravery~~ can get in O(1) use selection sort, take $O(n)$ time

(d) magic lineage $3\lceil \lg n \rceil + 4$? ⇒ the computing every student's ~~ances~~ magical lineage first then take counting sort ⇒ O(n) radix sort

Memo No.
Date / /

Problem 6. Triple Sum.

A, B, C. n integers, $O(n^2)$-time

$a+b+c=0$

containing n integers

means $a+b=-c$

1. first we can iterate C and make C's set C's items to be inverse. if +, means - + inverse,
2. then, ~~the set two level loop, iterate~~ restore C into a hash map, which can be used number in amortized time $O(1)$
3. iterate A and iterate B inner loop, get every i_A+i_B and check in C's map, the worst case of runtime is $O(n^2)$, check in map take $O(1)$ time amortized. then if checked return i_A, i_B and i_C, otherwise return false.

expected

Mo Tu We Th Fr Sa Su

Memo No. _____

Date / /

Problem). Where am I?

AVL Tree T containing n nodes

v describe O(logn)-time return index i of node v in the traversal order of T

u.height & u.size

1. first we can find the root of the tree T. take O(log n) time in the worst-case,

2. construct a ~~func~~ recursive function to find node v if in root.left or root.right, the function like:, and new two item A, B

```
def find_v_left_or_right(node):
    if node == v: return true
    ~~if node == v: return true~~
    ~~else~~ if node == None: return false
    A = find_v_left_or_right(node.left)
    B = find_v_left_or_right(node.right)
```

before that check if root = v, OR if root = v, index = size(v.left) ↑ return index

the ~~use~~ call the function with root, then three cases:

1. A = false & B = false, means v is not

Mo Tu We Th Fr Sa Su

Memo No. _____

Date / /

In the tree

2. A = ~~true~~ true: means v is in the left subtree of root, then take a judge of two case:

1. if v.parent.left == v: i = v.size - 1

if v.parent.right == v: i = v.parent.left.size + 1

3. B = ~~true~~ true, means v is in the right subtree of root, then take a judge of 2 cases:

1. if v.parent.left == v: i = root.left.size + v.size

2. if v.parent.right == v: i = root.left.size + v.parent.left.size + 1

4) all take O(1) time

because the height of a tree is log n, the find_left or height method recursion's back take in O(log n) time in worst case.

then the runtime of whole Algorithm is O(log n) ✓

or we can compute counts to reach root node.

Mo Tu We Th Fr Sa Su

Memo No. _____

Date / /

Problem 8.

n pipes (p_i, d_i)

p_i: pipes d_i: the distance of the hole from the front of pipe p_i.

- if each one has most one hole, patch any one
- else patch $|d_j - d_i|$ smallest pipe

1. Initialize(H) $H = \{(p_0, d_0), \cdots, (p_{n-1}, d_{n-1})\}$ in $O(n)$
2. report (p_i, d_i) $O(\log k)$ time
3. patch() follows the priority scheme above in $O(\log k)$ time ← (findMax)

k: the number of unpatched holes in the network

idea)

chose binary-Heap to store (p_i, d_j) which as a priority-list which p_i's $|d_i - d_j|$ is smallest if p_i has more than one hole to support patch()

Mo Tu We Th Fr Sa Su

Memo No. _____

Date / /

Anel A hash-map store every P_i, key is a linked list store d_i in P_i.

1. initialize (H): Creat a empty hash map H

when input (P_i, d_i). 1. if H(P_i) is not exist, then add P_i ~~to hash map's key~~, and construct ~~d_i as d_i.item = d_i, and d_i.next = a linked list~~ ✗

~~a AVL tree as a binary search tree~~

a binary-heap to store d_i as hash table's value.

Insert ~~int~~ d_i into binary-heap's as root.

2. if H(P_i) exist, add d_i into the binary-heap of hash table value: add into array's end and do max_heapify_up(A, d_i) → almost O(log k)

⇒ O(n + n log k) = O(n) expected

2. report (P_i, d_i): use hash-table to find P_i. O(1) amortized then find d_i is binary-heap : O(log k) ⇒ O(log k) expected

3. patch(): back to build, when build, we construct a array to store $|d_i - d_j|$ when build, and every item pointer it's pipe. 1. Use binary search to find min of array, then get

Mo Tu We Th Fr Sa Su

Memo No. _____

Date / /

p_i, ⇒ then find binary-heap of d_i in hash table,

delete ~~X~~

not work!

plus = answer!

idea: use AVL tree to store d_i, hash map store p_i and tree, the ~~sub-tree augmention~~ main | $|d_i - d_j|$ min is not $d_i - d_i.left$ or $d_i - d_j.left$ which can be find in O(1) time

when build ⇒ store in array ? ⇒ see answer

{ 1. A set AVL Tree T_p for each pipe p containing all the unpatched holes in p keyed by hole distance

2. A Hash Table D mapping p to its tree T_p

3. A binary-min-heap Q containing each consecutive pair of holes (p, d_1, d_2) with key of distance $|d_2 - d_1|$ and any holes one pipe with key ∞.

4. a hash map table mapping pair (p, d_1, d_2) to their location in Q

Mo Tu We Th Fr Sa Su

Memo No. _____

Date / /

Problem 9. Vaper Invite

n users. user Id updatable status, active or inactive

(a, b) ⇒ $d_i \in (a, b)$ must be active

want most

build(D) $D = \{d_0 \cdots d_{n-1}\}$ all active $O(n \log n)$

toggle_status(d_i) $O(\log n)$

biggte_active_range() Return (a, b) containing the largest number of actve users possible.

we store users in two AVL tree, one is active tree, another is inactive tree, all tree store user's Id in traversal order.

1. build(D), init a empty AVL tree, then built users as a AVL tree in $O(n \log n)$ time ✓ the ids in traversal order. maintain the min id in subtree

2. toggle_status(d_i) find d_i in AVL tree active, delete it in traversal order, rebalance and insert into inactive AVL tree

Mo Tu We Th Fr Sa Su

Memo No.

Date / /

all position take O(log n) time worstest case expected ✓

3. big-active-range(): ~~just return root.id as b~~

2 case, ~~1. root.id as — root.left.id root.left.ic~~

we main the size of subtree in AVL tree,

~~just compare root.left.size and root.right.size~~

~~if left.size bigger, return root.id as b, root~~

(we maintain max id & ~~smallest~~ minimun id as subtree augmention, ~~every time change~~)

2. supplement toggle-status(di) negative to nertive same as nertive to negative O(log n) time,

~~then if root.left.size > root.right.size, return root.id as~~

return root.right.max_id as b. root.~~big~~left-min-id as a, return a, b in O(1) time. ✓

Mo Tu We Th Fr Sa Su

Memo No. _____

Date / /

Answer of Problem 9.

maintain a single set AVL Tree T storing each user ID and their status, keyed in ID. augment each node x in T with four subtree properties.

① x.size

② x.suffix = (d, m), the smallest id of the subtree is active

③ x.prefix = (d, m), the largest id of active user

④ x.substring = (a, b, m), a, b are IDs from the subtree where each of the m IDs d in the subtree with $a \le d \le b$ is active and m is maximized