Lecture24

Monday, 29 May 2023 10:00

hashmap implementation } internal hashmap

put
get
Contains key
Remove } → O(1) time

Key - Value pair
Keys are unique
Values can be duplicated

linkedlist <HTPair> [] bucketarray = new linkedlist[capacity];

class HashMap<K, V>
{
K key;
V value;
HTPair(K key, V value)
{
this.key = key;
this.value = value;
}
}

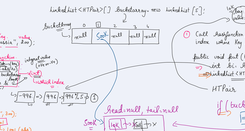
HTPair p = new HTPair(2, 3);

linkedlist<Student> stlist = new linkedlist<>();

put(K key, V value) {

int bucketIndex = getBucketIndex(key);
int ht = findHt(key, bucketIndex);
if (ht == -1) {
HTPair pair = new HTPair(key, value);
bucketarray[bucketIndex].add(pair);
size++;
}
else {
HTPair pair = bucketarray[bucketIndex].get(ht);
pair.value = value;
}
}

linkedlist<HTPair>[] bucketarray = new linkedlist[5];



① Call hashfunction, get the hash index value key hash to add

public void put(K key, V value) {
int bucketIndex = getBucketIndex(key); ⇒ O(1) ①
LinkedList<HTPair> bucket = bucketarray[bucketIndex]; // null or address
HTPair pair = new HTPair(key, value);
if (bucket == null) {
bucket = new linked list <>();
bucket.add(last, pair);
this.size++;
bucketarray[bucketIndex] = bucket;
} else {
int findHt = bucket.find(pair.key);
if (findHt == -1) { // Index is not present
bucket.add(last, pair); ⇒ O(1)
this.size++;
} else {
HTPair pair = bucket.get(findHt);
pair.value = value;
}
}

Set
key, Index
value, Int

} O(1)

Return null;

get(K key) {
remove(K key) {

}

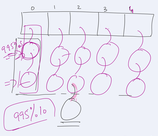
put
get
Remove } ⇒ O(n) time
↳ findHt

find → O(n) time

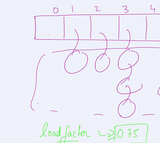
double loada = (this.size + 1) / bucketarray.length;
if (loada > 2) {
this.rehash();
}

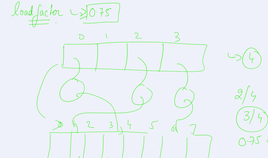
before

1/5, 2/5, 3/5, 4/5
5/5, 6/5, 7/5
8/5, 9/5, 10/5
11/5 > 2

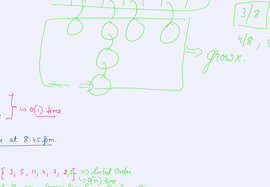


⇐ avg Case ↳ O(2)





load factor → 0.75



2/4
3/4 ≥ 0.75
0.75 == 0.75
3/8
4/8, 5/8
6/8 ≥ 0.75

⇒ grows.

put
get
Remove } → O(1) time

Resume at 8:15 pm.

Q) arr: { 3, 5, 11, 4, 1, 2, 6 } → Sorted Order
↳ O(n) time
Length of the max consecutive sequence from this array?

3, 4, 5 , 1, 2, 3 , 1, 2
Ans ⇒ 4 ⇒ 3 ⇒ 2

naive ⇒ Sort → nlogn →
{ 3, 5, 11, 4, 1, 2, 6 } → O(n)

Integer	Boolean
3	true
5	true
11	true
4	false
1	true
2	true
6	false

max = arr[0]
while (arr[i] == arr[i-1] + 1) {
max = arr[i] + 1;
i++;
}

max 3 ⇒
max = 3 ⇒
max = 5 ⇒

max 4 ⇒
max = 3 ⇒
max = 5 ⇒

map is ready

int maxlen = 0;
int arrlastkeyValue = 0;
for (—) {
if (arrlastkeyGetValue == false) {
continue;
}
int currlastkeyLen = 0, currlastkeyStartValue = arrlastkeyget
while (map.containsKey(currlastkeyStartValue + currlastkeyLen)) {
currlastkeyLen++;
}
if (currlastkeyLen > maxlastkeyLen) {
maxlastkeyLen = currlastkeyLen;
}
}

① → 1
② → 3
③ → 4
④ → 5

max = len — .

↳ O(n) + O(n) → O(n)

check subarray with sum zero

Q → ~~t~~ → O(n)

Q → maxlen of subarray with sum zero?

{ 3, 1, 1, 2, 5, -5, -2 }
0 → 2
↳ 4

len = 6 ⇒ 6

	3	-1	1	2	5	-5	-2
prefsum	3	2	3	5	10	5	3
index	0	1	2	3	4	5	6

Integer	Integer
3	0
2	1
5	3
10	4

→ first occurence of my prefix sum.

len = j - i

2 - 0 ⇒ 2

len = j - i ⇒ len

5 - 3 = 2 =

6 - 0 = 6

(i+1, j) → subarray has sum zero

3, ..., 3, ..., 3, ..
6 11

7-1 < k-1 | <<jck

hashmap
↳ unordered | TreeMap