DATA STRUCTURES AND ALGORITHMS

INDEX

NAME. : Smit Sekhadia Std. : Div. :

Roll No. : School / College. :

Sr. No	TITLE	Date	Teacher's Sign / Remark
1	LINKED LIST.		
2	STACKS		
3	QUEUES		

Page No.
Date

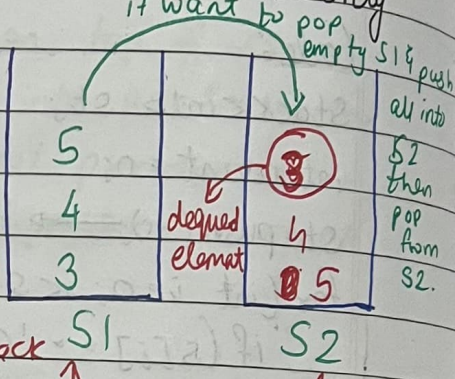
QUEUES

Q22] Implement Queue using Stacks S1 & S2.

→ You have 2 stacks.
→ For Enqueue push data into S1
→ For dequeue, pop all the data from S1 and push it into S2 till S1 is empty.
→ Now pop from S2 to dequeue.
→ Now again if you want to insert; (enqueue) Push in S1 only

Code:

```
void push(int x)
{ S1.push(x); }
int pop(){
    if(S2.size()==0){
        if(S1.size()==0){
            return -1;
        }else{
        while(S1.size()==0){
            int a = S1.top;
            S1.pop();
            S2.push(a);
        }
    }
    int b=S2.top();
    S2.pop();
    return b;
}
```



if both Stack empty

pop all elements from S1 and push into S2.

Time: O(1) → Push
O(n) → Pop

Space: O(2N)
Here we are not using any extra space. Stacks are already given to us.

Page No.
Date

Q19] Implement Stack using 2 Queues.

→ Push data into q_1

→ ~~Empty~~ put all the elements of q_2 into q_1 untill q_2 is empty.

→ Swap elements of q_1 & q_2 & repeat same operation for push

→ To Pop; just pop out the front element of q_2.

Code:

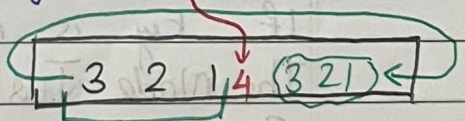
```
void Push (int x) {
  q1.push (x);
  while (!q2.empty()){
      q1.push (q2.front());
      q2.pop();
    }
  Swap (q1, q2);
}

int Pop (){
  int ans;
  if (q2.empty()) {
    return -1;
  }
  else {
    ans = q2.front();
    q2.pop();
    return ans;
  }
}
```

Time: O(n) → To push; O(1) → To pop.

Space: O(2n) → 2 Queues.

Using 1 Queue:

maintain size variable. We will pop size-1 elements from front of q & push them back. i.e eg: in queue there are 3 2 1 already; Push(4)



pop size-1 elements & push them behind 4

```
push (int x){
  q1.push(x);
  int sz;
  sz = q1.size()-1;
  while (sz--){
      q1.push (q1.front());
      q1.pop();
    }
}
```

Pop will be same as above → pop the first element of q1.

Time → O(n) → To push
O(1) → To Pop

Space → O(n) → 1 queue used.

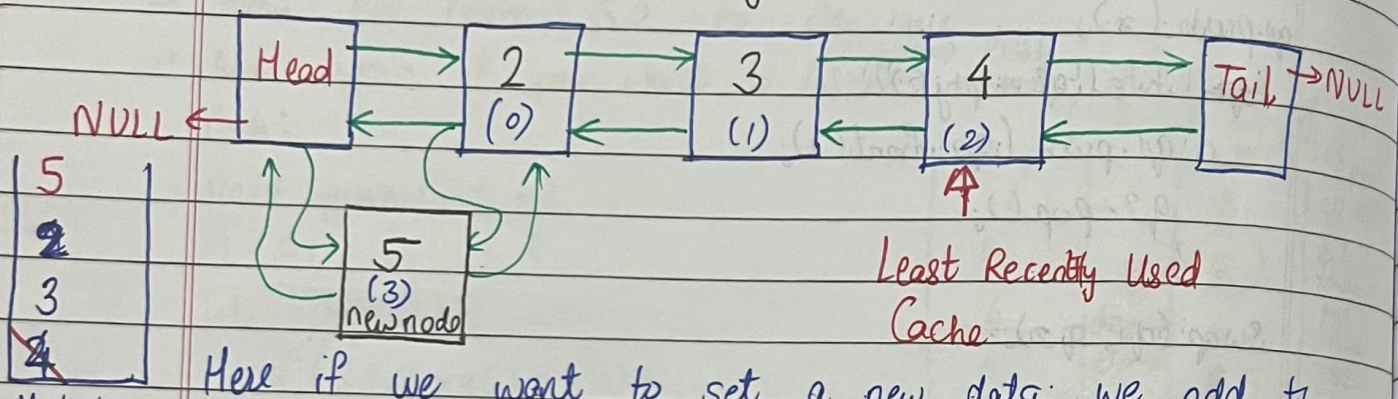
DSA

Q25) LRU Cache. (Good Ques)

→ We will use Hash Map and Queues using Doubly Link list. When we have to set a cache; we add its key to the hash Map and value will be Node of DLL which will contain the key and value of cache.

→ Why we use Doubly Linked List?

Consider; hash map capacity = 3



Here if we want to set a new data; we add the temp node just before head, & if the cache is already full; you remove (delete) the LRU node i.e 4

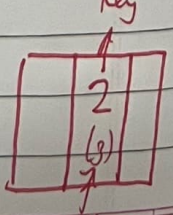
If we want to get data, first we find the key is present in Hashmap or not. If present → remove key from map; delete that Node, add Node to DLL just after head; and now add the key to the map as heads→next. If key is not present in map; return -1.

The Node Just Before Tail will always be LRU.

Code:

```
Class Node {
  int key;
  int value;
  Node * next;
  Node * prev;
  Node (int key, int value) {
    this → key = key;
    this → value = value;
  }
};
```

} Storing key & its value of cache in Node



```
Node* head = new Node(-1, -1);
Node* tail = new Node(-1, -1);
int cap = 0;
unordered_map<int, Node*> m;

LRU Cache(int cap){
  this->cap = cap;
  head->next = tail;
  tail->prev = head
  head->prev=; tail->next = NULL;
}
void addNode(Node* newnode)
{ Node* temp;
  temp = head->next;
  newnode->next = temp;
  newnode->prev = head;
  head->next = newnode;
  temp->prev = newnode;
}
```

Adding new node just after head.

```
void deleteNode(Node* delnode){
  Node* delprev;
  Node* delnext;
  delprev = delnode->prev;
  delnext = delnode->next;
  delprev->next = delnext;
  delnext->prev = delprev;
}
```

Deleting the node.

Time: O(n) for both get & set as we use Maps.
Space: O(1) for both get & set.

Page No.
Date

```
int get(int key){
    if(m.find[key] != m.end()){
        Node* resnode = m[key];
        int res = resnode -> value;
        m.erase[key];
        deletenode(resnode);
        addnode(resnode);
        m[key] = head -> next;
        return res;
    }
    else{
        return -1;
    }
}
void set(int key, int value){
    if(m.find(key) != m.end()){
        Node* existingnode = m[key];
        m.erase(key);
        deletenode(existingnode);
    }
    if(m.size() == cap){
        m.erase[tail -> prev -> key);
        deletenode(tail -> prev);
    }
    addnode(new Node(key, value));
    m[key] = head -> next;
}
};
```

if you find the key in the hashmap before the map ends

Node* resnode = m[key]; → Store the key in resnode

delete that node

add that node just before head

Add the key of that node in map.

if key not found in map; return -1

if you find the key in the hashmap before map ends

erase the key if key is found

as we need to update value for that key.

delete that node.

if Hashmap is full

Erase the LRU key

delete LRU node.

add the new node which we want to Set

add the data in the map.

Time: Enqueue → O(1)
Dequeue → O(1)

Space: O(n)

Page No.
Date

[Q23] Implement k Queues in Array Efficiently. (Good Ques)

→ Similar to k stacks in Array.
→ arr[n] → store Elements of k Queues.
→ front[k] → store index of first element of k^{th} Queue.
→ rear[k] → store index of last element of k^{th} Queue.
→ next[n] → ① If queue index is empty store the next free index
② if index is occupied, store the index of the prev element of that queue.
→ free variable → that will contain the next free index.

Code:

```
Enqueue(int data, int k){
  if(isFull()){
    cout << "Queue overflow";
    return;}
int i = free; → i will store free available space index
free = next[i]; → free → next available space index
if(isEmpty()){ → if kth Q is empty.
  front[k] = i; first element will be front
}else{ → Else update rear with a element's
  next[rear[k]] = i; before's index
next[i] = -1;
rear[k] = i; update rear as new inserted data
arr[i] = data;
}        → add data to Q in array.
```

```
Dequeue(int k){
  if(isEmpty(k)){
    cout << "Queue Underflow";
    return;
  }
  → as you have to dequeue
int i = front[k]; first element of kth Q
front[k] = next[i]; as we are storing index of prev element of each element if the index is already occupied. So this will be front
next[i] = free; → Attach prev front
free = i; to begning of free list
  → return the prev front item
return arr[i]; from array.
}
```

```
Constructor: KQueues(int k1, int n1){
{ k = k1, n = n1;
  arr = new int[n];
  front = new int[k];
  rear = new int[k];
  next = new int[n];
  for(int i=0; i<k; i++){
    front[i] = -1;}
  free = 0;
  for(int i=0; i<n-1; i++)
    {next[i] = i+1;}
  next[n-1] = -1;
```

DSA450

Q26] Reverse Queue Using Recursion

→ Pop the front of Queue & store it in data var
→ Call the recursion again on that Queue.
→ push the data in Queue (will be call after recurssion time)

Code:

```
Queue<int> rev (queue<int> q){
  if (q.size()!=0){
    int data = q.front;
    q.pop();
    q = rev(q);
    q.push(data);
    return q;
  }
  else return q;
}
```

Problem in this is that in recurssion stack each time entire q will be stored making Time & space complexity more
Solution: Make another function & pass queue by ADDRESS (&q)

Code:

```
Que<int> rev (que<int> q){

    func(q);
    return q;
}
void func (queue<int> &q){
    if(q.size()!=0){
      int data = q.front();
      q.pop();
      func(q);
      q.push(data);
    }
}
```

V.VIMP Remember for ahead Codes also

→ Passing by address so that that same q is updated on whatever operation we do; else everytime a new queue would be made in the memory.

Page No.
Date

Q26] Queue & Stack using Dequeue.

→ Make a Doubly Linked List.
→ make 4 functns: insertfirst(); deletefirst(); insertlast(); deletelast();
→ These are normal Linked List Functns
→ Make a ~~Front~~ Stack which will have
class
- push → insertlast()
- pop → deletelast()

→ Make a class Queue which will have
- Enqueue → insertlast()
- Dequeue → ~~remove~~ delete first()

Q27] Reverse First K elements of Queue.

↳ Only first K elements; rest elements as it is.

→ Make a stack s; pop the first k elements from ~~stack~~ queue and push it into stack.
→ pop the elements and push them in queue
→ Now the first 'k' elements are reversed. but they are behind of the rest of elements.
→ Pop the remaining elements and push them again in queue to get the proper order.

Code:

```
stack<int> s;
for(int i=0; i<k; i++){
    int data = q.front();
    q.pop()
    s.push(data);
}
while(!s.empty()){
    int data = s.top();
    s.pop();
    q.push(data);
}
```

1 2 3 4 5 K = 3

4 5 3 2 1 ← pop & push the remaining element to back of queue.

∴ 3 2 1 4 5

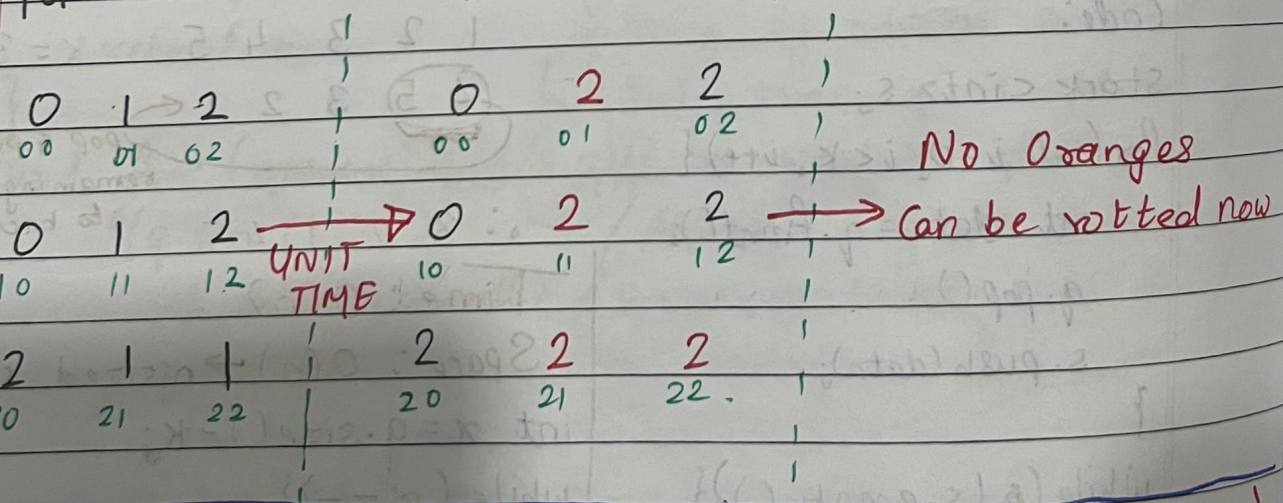
Time: O(n)
Space: O(n) → n sized stack used.

```
int x = q.size() - k;
while(x--){
    int data = q.front();
    q.pop();
    q.push(data);
}
return q;
```

DSA450

30] Rotten Oranges.

→ make a pair <int, int> to store indexes of element.
→ Scan the matrix wherever 2 is found, push them in queue; set count as 0.
→ Push {-1, -1} in the stack, {-1, -1} acts as delimeter (Seperator). iss se pata chalega ki ek hr ho gaya and vo orange se baki ke orange rot hue
→ While queue is empty, check if first element of queue is {-1, -1} & size is 1; then pop & break else check if first is {-1, -1} and size of q is > 1 then increase count as aur orange rot hone baki hai.. pop the element {-1, -1} and push it back at end of queue again {-1, -1} ko so we know unit time me sab oranges rot hue hai & we check again rot aur ho sakte hai ki nai.
→ Else now check if there is some element which can be rotted (jo healthy) [up, down, left, right] ?. if yes, push it into queue and make its value 2.
→ pop elements till queue is empty.



(0,2), (1,2), (2,0), (-1,-1), (0,1), (1,1), (2,1)(2,2) (-1,-1)

pop pop pop now push possible nos.

push all element with index 2

Now pop (-1,-1) & increase count by 1 & push (-1,-1) again

Then pop these elements & repeat process.

Q32] First negative integer in K size window.

Sliding Window + queues

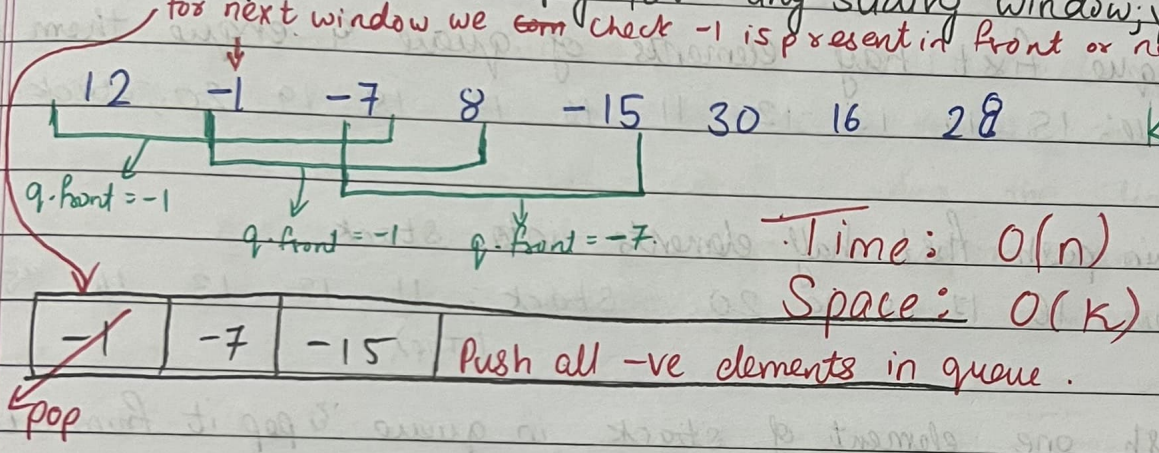
→ Make a ans vector.

→ Make a sliding window of size K. & push all the negative numbers in queue.

→ for each increase in window, ans will be front of the queue

→ Compare the first element of queue with front; if it is present then pop that element as in the next window; that first element will be removed.

→ If the queue is empty for any sliding window; you return 0



Time: O(n)

Space: O(K)

Fixed Code in ALL SLIDING WINDOWS OF FIXED SIZE

```
while (j < size)
{
   Calculations ——→ vary acg to problem

   if ( j-i+1 < K) {
      j++;
   }
   if ( j-i+1 == K) {
   ① ans nikalo from Calculation → Vary acg to problem
   ② Slide window ahead
   }   ↳ before slide window, remove calc performed for
   }       i as it will not be part of next window.
```

Q28] Interleave first half of Queue with second half.

Queue: 11 12 13 14 15 16 17 18 19 20.

→ Output: 11 16 12 17 13 18 14 19 15 20.

→ Push first half elements of Queue into stack and pop from queue.

queue: 16 17 18 19 20 stack: 15 14 13 12 11 (TOP)

→ Enqueue back the stack elements till stack is empty.

queue: 16 17 18 19 20 15 14 13 12 11; Stack: Empt

→ Deque first half elements of queue & enque them back

queue: 15 14 13 12 11 16 17 18 19 20; Stack: Empt

→ Again push first half elements into stack.

queue: 16 17 18 19 20 stack: 11 12 13 14 15 (TOP)

→ push one element of stack in queue & pop it from stack
→ Push first element of queue in queue again at back of queue
→ pop that element of queue from front
→ do till stack is empty.

quae: 11 16 12 17 13 18 14 19 15 20.

```
while (!s.empty()) {
    q.push(s.top());
    s.pop();
    q.push(q.front());
    q.pop();
}
```

Time: O(n)

Space: O(n)

Page No.
Date

Q] Maximum element in Subarray of sliding window size K.

Sliding window + Queues.

Brute force

↳ 2 for loops where i goes from 0 to n

↳ j goes from 0 to i+K (window size)

↳ ~~Check~~ Find maximum of that ~~elements~~ window & push in vector

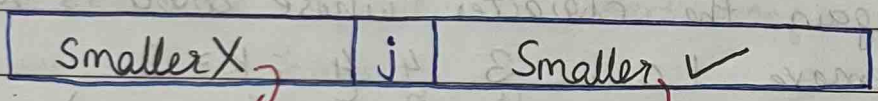
Optimised Solution: (Sliding windows & Queues)

→ Start i & j from 0

Till you hit ~~fate~~ sliding window: Calculations →

push first element, and for the next elements; check if front is less than pop front & push that element. now till the sliding window is hit; push elements ahead into queue even if they are small as when the sliding window moves ahead they might be ~~g~~ maximum for that window

If you find a greater element pop all the elements in front of that



Time: O(n*k)

Space: O(n)

→ ans ka calculation → Whenever window size hits; ans = q.front()

→ to remove calculations of i as in next window i will not be there; check if i is q.front(): if yes pop front.

```
vector<int> ans;
int i=0, j=0;
Dequeue<int> dq;
while (j < size){
    if (dq.empty()){
        dq.push(arr[j]);
    }else{
        while(!dq.empty() && q.back()<arr[j]){
            dq.pop_back()
        }
        dq.push_back(arr[j]);
    }
    if (j-i+1<k){ j++; } → increase till sliding win hits
    else if (j-i+1==k){ → sliding window hit
        ans.push_back(dq.front()); → ans calc
        if (arr[i]==q.front());
            dq.pop_front(); → remove calc of i as in next window it will not be there.
        i++; } slide window
        j++;
    }
```

Calc until sliding window is hit

if prev element are < arr[j]

DSA450 ★★★

Q34] Sum of minimum and maximum of all subarrays of size k.

→ do same as prev sum (& find max element in subarray)
→ do same to find minimum [use 2 deque one for Max & other for Min]
→ Add both the output vectors to get sum.

DSA450.

Q35] Minimum Sum of Squares of character counts in a given string after removing 'k' characters.

String: aaabbbbccccc K=3

3 4 5

Here you have to get min sum of square after removing 3 characters.

Case ① You remove 3 c's then $3^2+4^2+2^2 = 29$ ✗

Case ② You remove 2 c's & 1 b; $3^2+3^2+3^2 = 27$ ✓ minimum

Case ③ You remove 2 b's & 1 c; $3^2+2^2+4^2 = 29$ ✗

You remove character whose count is max; decrease k remove again the character whose count is max.

→ First remove 1 c ∴ 3 4 4 ; K = 2
a b c

→ count(b) = count(c) ∴ remove any one (suppose b); 3 3 4 ; k=1
a b c

→ count(c) is max ∴ remove c; 3 3 3 ∴ $3^2+3^2+3^2 = 27$; k=0
a b c

→ Use priority Queues (or (Max Heap)) whose front will always be the maximum element.

→ Use unordered map to know the count

→ pop the max element from pq & decrease its count by 1 & push it back again in pq. & decrease K by 1

→ Continue till K=0.

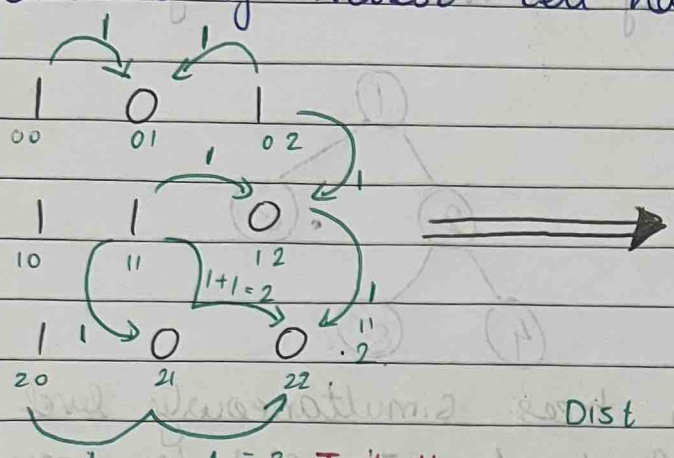
Time to push data in pq is log n
& it runs for map of max size n ∴ n log n

Time: O(n log n)

Space: O(n) → 1 priority queue used

Page No.
Date

Q31] Distance of Nearest cell having 1 in Binary Matrix.



0 1 0
0 0 1
0 1 2.

Dist of 1 from 1 will be 0 only.

Initially ans matrix will look like

0 M 0
0 0 M
0 M M

In solution instead of checking dist from 0's to 1's check dist from one to 0's. and store them in ans

→ Start from (0,0); on its right is INT_MAX now check.
if a[i][j] + 1 < INT_MAX; 1 < INTMAX → True;
update it with a[i][j+1] = a[i][j]+1. & push it in the back of queue

→ Check below (0,0); it is already 0 ~~→ No need to compare~~ 0+1 < 1 → not true so move ahead.

→ Initially You Have To push all elements with 1 in the ques into the queue to check; aage kin elements ko compare karna hai.

→ as now work of (0,0) is completed; pop it from queue.

→ Next element in queue is (0,2)

→ see on left of (0,2); you have 1 (already got from above) check if 1+1 < 1 ∴ False so let that remain as 1.

→ see down; there is INT_MAX; check if a[i][j]+1 < INTMAX
True then update that INTMAX to 1 & push it (1,2) into queue

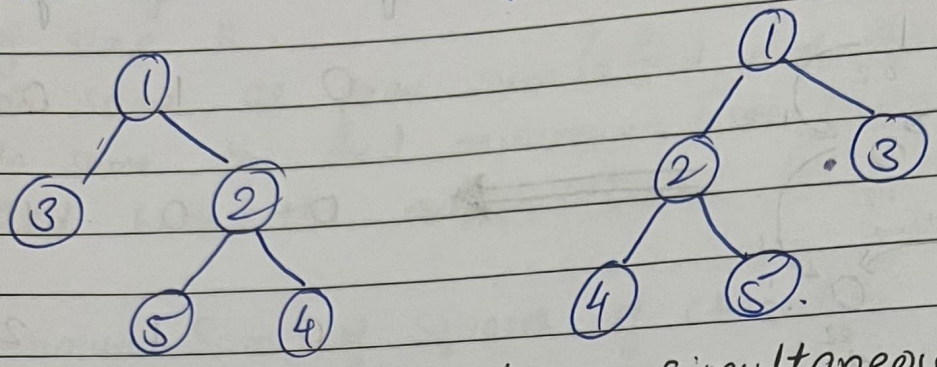
→ Pop (0,2) out of queue.

→ Coming to (1,2) our matrix will look like: 0 1 0
& queue would be: (1,2), (2,1) 0 0 (1) a[i][j]
i=1; j=2 a[i][j]+1 < INT_MAX ⇒ True 0 1 INT_MAX.
∴ a[i+1][j] (22). a[i+1][j]

∴ a[i+1][j] = a[i][j]+1 ∴ 0 1 0
= 1+1 0 0 1
a[i+1][j] = 2 0 1 2

Date

Q] Check if all levels of two trees are anagrams or not



We Traverse both trees simultaneously level by level
We store each level elements of both trees in 2 different vectors.
To check if two vectors are anagram or not; Sort both and then compare the two vectors.
If they are same They are Anagrams.
Time : $O(n \log n)$; $n \rightarrow$ number of nodes.