Advanced Problem Solving on Queues

Relevel
by Unacademy

List of Problems Involved

- Arrays Permutation
- Ice Cream Purchase
- Circular Tour
- Average of Levels
- Negative Integer in Window

Arrays Permutation

Stack Permutation - A Stack Permutation is defined as converting an array to another array such that we can only apply Stack Operations on the input array.
Problem Statement - Given two queues. Your task is to check if we can convert one queue to another queue using Stacks permutation.
Below rules can only be followed -

- The dequeue operation can be applied on the input queue
- Push and Pop operations of Stack
- Stack used and Input queue must be empty at the end
- Enqueue is allowed to the output queue

Input - [4,5,6], [5,4,6]

Output - True

Arrays Permutation

Approach – Since we need to check whether given queues are stack permutations, we will use stack and perform queue operations on the input queue.

Steps -

1. Dequeue elements from the input queue and check whether the element is equal to top of the output queue
2. If elements are not equal, push to stack
3. If elements are equal, dequeue elements from both queues and compare the top of the stack with the top of the output queue. If they are equal, pop elements from the stack and output queue. If they are not, repeat step 1
4. Repeat the above steps till the input queue and stack are empty.
5. If both are empty, print True else False

Arrays Permutation

Time Complexity –

If the number of elements in the input queue is n then, the time complexity will be O(n)

Space Complexity –

If the number of elements in the input queue is n then, the space complexity will be O(n)

Code Link -> https://jsfiddle.net/u7y1oc38/

Arrays Permutation

```
function stackPermutation(ip, op, n) {
  let input = [];
  for (let i = 0; i < n; i++) {
    input.push(ip[i]);
  }
  let output = [];
  for (let i = 0; i < n; i++) {
    output.push(op[i]);
  }
  let tempStack = [];
  while (input.length != 0) {
    let ele = input.shift();
    if (ele == output[0]) {
      output.shift();
      while (tempStack.length != 0) {
        if (tempStack[tempStack.length - 1] == output[0]) {
          tempStack.pop();
          output.shift();
        } else
          break;
      }
    } else {
      tempStack.push(ele);
    }
  }
  return (input.length == 0 && tempStack.length == 0);
}
```

Ice Cream Purchase

Problem Statement - Given a queue of N persons in the form of an array. Each element in the array denotes the currency of note that a person has. These persons are waiting to purchase an ice cream from X which costs Rs 5. Possible currency of notes can be 5, 10, and 20.
Initially, X has 0 amount. Your task is whether X can provide change to each person or not.

Input - [5,5,10,20]

Output - Yes

Explanation -
When a third person will come, X will have 2 notes of 5. X will give him one note of 5. Then the fourth person will come. Now X can provide him with one note of 10 and one note of 5 which is 15 total.

Ice Cream Purchase

Approach – Since, we need to check whether X person can provide change, we will keep track of currency notes of 5 and 10 only. Because 20 is the highest currency note and it cannot be provided as a change.

Also, we will use a queue here to keep track of given input elements.

Steps -

1. Initialize 2 variables - countFive and countTen
2. If the person has currency note of 10, and countFive > 0, then countFive- - and countTen++
3. If X does not have 5 currency notes then n change can be provided
4. If a person has 5 currency notes, then fiveCount++
5. If person has 20 currency note, then below steps will be followed -
    - If countFive > 0 and countTen > 0, decrease both.
    - Else if, countFive >= 3, decrease fivecount by three.
    - Else - return false.
6. If all persons get change, print Yes else No

Ice Cream Purchase

Code Link -> https://jsfiddle.net/pvstfjn9/

Time Complexity –

If the number of elements in the input queue is n then, the time complexity will be O(n)

Space Complexity –

If the number of elements in the input queue is n then, the space complexity will be O(1)

Ice Cream Purchase

```
function change(notes, n) {

  let countFive = 0;
  let countTen = 0;

  for (let i = 0; i < n; i++) {
    if (notes[i] == 5)
      countFive++;
    else if (notes[i] == 10) {

      if (countFive > 0) {
        countFive--;
        countTen++;
      } else
        return 0;
    } else {
      if (countFive > 0 &&
        countTen > 0) {
        countFive--;
        countTen--;
      } else if (countFive >= 3) {
        countFive -= 3;
      } else
        return 0;
    } }
  return 1;
}
```

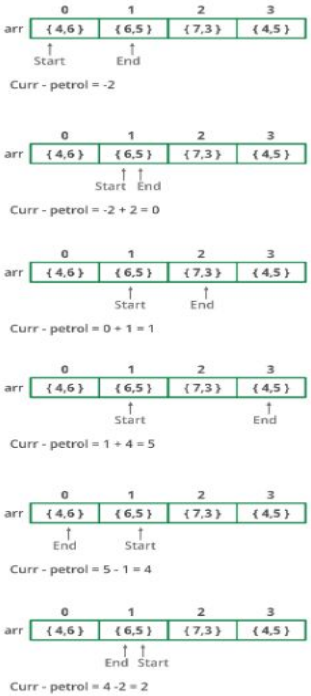
Circular Tour

Problem Statement - Given a circle where many petrol pumps are installed. Also, the amount of petrol at each petrol pump and the distance between them is given. Your task is to find the first point in the circle where a truck can complete its first circle.
Assume that truck can cover 1 unit of distance with 1 liter of petrol

Input - (4,6), (6,5), (7,3), (4,5)
Above example shows that there are 4 petrol pumps in a circle, where the first number indicates the amount of petrol and the second number indicates distance from the next petrol pump.
Output - (6,5)

Approach - We will use a queue here to implement this solution. We will enqueue the petrol pump till we have sufficient petrol to complete the tour. If we have negative petrol, then we will dequeue them till the amount of petrol becomes zero.

Circular Tour



Circular Tour

Steps -

1. Initialize two pointers start and end with 0 and 1
2. Curr_petrol = current pump petrol value - current pump distance value
3. Iterate through a loop till we reach our first petrol pump again with either 0 or more amount of petrol
4. Iterate till curr_petrol < 0 and start != end -> update start to the next and update curr_petrol
5. If start = 0, no solution
6. End loop step 4
7. Update curr_petrol
8. Update end pointer
9. End loop step 3
10. Return the start pointer

Circular Tour

Time Complexity –

If the number of petrol pumps is n, then the time complexity will be O(n)

Space Complexity –

If the number of petrol pumps is n, then the space complexity will be O(1)

Code Link - https://jsfiddle.net/1xchrn59/

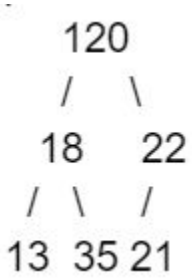
Circular Tour

```
class petrolStation {
  constructor(petrol, distance) {
    this.petrol = petrol;
    this.distance = distance;
  }
};

const print = (arr, n) => {

  let start = 0;
  let end = 1;

  let curr_petrol = arr[start].petrol - arr[start].distance;

  while (end != start || curr_petrol < 0) {

    while (curr_petrol < 0 && start != end) {
      curr_petrol -= arr[start].petrol - arr[start].distance;
      start = (start + 1) % n;

      if (start == 0)
        return -1;
    }

    curr_petrol += arr[end].petrol - arr[end].distance;

    end = (end + 1) % n;
  }

  return start;
}


let arr = [new petrolStation(6, 4), new petrolStation(3, 6), new petrolStation(7, 3)];
let n = arr.length;
let start = print(arr, n);

(start == -1) ? console.log("No solution exists!!") : console.log(`Start = ${start}`);
```

Average of Levels

Problem Statement - Given a binary tree. Your task is to print the average of the nodes present on each level.

Input -



Output -

Level 0 - 120

Level 1 - 18 + 22 = 40/2 = 20

Level 2 - 13 + 35 + 21 = 69/2 = 23

Average of Levels

Approach – Since we need to find the sum of nodes at each level, we can use level order traversal of the tree which can be done using a queue.

Steps -

1. Push a node to the queue and dequeue the node
2. When we dequeue the element, enqueue all its children to the temp queue
3. Repeat steps 1 and 2 till the queue is empty
4. Whenever the queue is empty, it shows that we have completed one level
5. Keep track of elements enqueued into temp queue and calculate their average
6. Continue the process till both queues become empty

Average of Levels

Code Link -> https://jsfiddle.net/0ydpxtkv/

Time Complexity –

If the number of nodes in the input tree is n then, the time complexity will be O(n)

Space Complexity –

If the number of nodes in the input tree is n then, the space complexity will be O(n)

Average of Levels

```
function average(root) {
  let q = [];
  q.push(root);
  let sum = 0,
    count = 0;
  while (q.length > 0) {
    sum = 0;
    count = 0;
    let temp = [];
    while (q.length > 0) {
      let n = q[0];
      q.shift();
      sum += n.val;
      count++;
      if (n.left != null)
        temp.push(n.left);
      if (n.right != null)
        temp.push(n.right);}
    q = temp;
    console.log((sum * 1.0 / count) + " ");
  }
}
```

Negative Integer in Window

Problem Statement - Given an array and an integer k. Your task is to find the first negative integer in a window of size k. If windows do not contain any negative integer, print 0.

Input - [-2,0,-3,-4], k = 2

Output - -2, -3, - 3

Approach – Since we need to find the first negative integer from the window of size k, we will use deque or a double-ended queue. We will store our required element in it which is the first negative integer in a window of size k.

Negative Integer in Window

Steps -

1. Initialize a double-ended queue
2. Iterate through the array and save negative integers into the deque
3. For a specific window, if the deque is not empty, then the front element of the deque will be the first integer of that window
4. Else no negative integer present

Time Complexity –

If the number of elements in the input array is n then, the time complexity will be O(n)

Space Complexity –

If the size of the window given is k then, the space complexity will be O(k)

Negative Integer in Window

Code Link -> https://jsfiddle.net/Lcj3mxva/

```
function firstNegativeInteger(arr, n, k) {
  var Di = [];
  var i;
  for (i = 0; i < k; i++)

    if (arr[i] < 0)
      Di.push(i);
  for (; i < n; i++) {
    if (Di.length !== 0)
      console.log(arr[Di[0]] + " ");

    else
      console.log("0" + " ");

    while ((Di.length !== 0) && Di[0] < (i - k + 1))
      Di.shift();
    if (arr[i] < 0)
      Di.push(i);
  }
  if (Di.length !== 0)
    console.log(arr[Di[0]] + " ");
  else
    console.log("0" + " ");
}
```

Thank You!