Problem Solving on Queues

Relevel
by Unacademy

List of Problems Involved

List of Problems Involved

- Binary Numbers Generation
- Queue using Stack
- Queue using Linked List
- Reverse First K Element from Queue
- Circular Tour

Binary Numbers Generation

Problem Statement

Given a number n. Your task is to generate binary numbers having decimal value from 1 to n

Input - 5

Output -

1, 10, 11, 100, 101

Approach – Our naive approach could be to use a simple loop from 1 to n and convert decimal to binary number simply.

We can use the queue to get an efficient solution. We will enqueue 1 to the queue. Then we will dequeue the first element to get the required element. Lets see each step one by one

Steps -

1) Create empty queue as q
2) Enqueue the first element which is 1
3) Iterate through a loop from 1 to n
4) Dequeue the front element and print it
5) Append “0” to the front element and enqueue it
6) Append “1” to the front element and enqueue it

Binary Numbers Generation

Time Complexity –
If given number is n then, time complexity will be O(n)

Code Link -> https://jsfiddle.net/96jsn1tq/

```
function generateBinary(n)
{
  var q = [];

  q.push("1");

  while (n--) {

    var s1 = q[0];
    q.shift();
    console.log( s1 );

    var s2 = s1;

    q.push(s1+"0");

    q.push(s2+"1");
  }
}

var n = 5;
generateBinary(n);
```

Queue using Stack

Problem Statement

Implement queue operations using the Stack

Approach – Our task is to implement Queue using Stack. Basically, we need to implement the below 2 operations -

1) Enqueue() - Adding element to rear
2) Dequeue() - Removing element from front

We will use 2 stacks to implement the above operations i.e. s1 and s2. We will make sure that whenever we call dequeue operation, it will pop the element from s1.

Dequeue operation fetches the element from the rear side and hence, we need to make sure that the oldest element will present at the top of stack s1. For this, we will use stack s2.

Steps -

1) Create 2 empty stack s1 and s2
2) enqueue(q, item) -
    - While s1 not empty, push elements from s1 to s2
    - Push item to s1
    - Push all popped elements again to s1
1) dequeue(q, item) -
    - If s1 empty, throw error
    - Pop item from s1 and return

Queue using Stack

Time Complexity –

Time complexity for enqueue() will be O(n)

Time complexity for dequeue() will be O(1)

Space Complexity –

Space complexity will be O(n)

Code Link ->https://jsfiddle.net/3vdwjhga/

```
class Queue {

  constructor() {
    this.s1 = [];
    this.s2 = [];
  }

  enQueue(x) {


    while (this.s1.length != 0) {
      this.s2.push(this.s1.pop());

    }


    this.s1.push(x);


    while (this.s2.length != 0) {
      this.s1.push(this.s2.pop());

    }
  }

  deQueue() {


    if (this.s1.length == 0) {
      document.write("Q is Empty");
    }


    let x = this.s1[this.s1.length - 1];
    this.s1.pop();
    return x;
  }
}
```

Queue using Linked List

Problem Statement

Implement queue operations using the Linked List

Approach – Our task is to implement Queue using Linked List. Basically, we need to implement the below 2 operations -

1) Enqueue() - Adding element to rear
2) Dequeue() - Removing element from front

Since its a linked list, we will add a new node at the last and update rear to its next node. Similarly, we will remove the front node and update the front next to it.

Steps -

1) enqueue(item) -
- Create a node
- If queue is empty, front and rear of queue will be node
- Else, append node at rear

2) dequeue() -
- Fetch element from front
- Update front to front.next
- If front = null -> rear = null

Queue using Linked List

Time Complexity –

Time complexity for enqueue() and dequeue() operations will be O(1)

Code Link ->https://jsfiddle.net/m74j1xfL/

```
function enqueue(key)
{

    let temp = new QNode(key);


    if (rear == null) {
      front = rear = temp;
      return;
    }


    rear.next = temp;
    rear = temp;
}


function dequeue()
{

    if (front == null)
      return;


    let temp = front;
    front = front.next;


    if (front == null)
      rear = null;
}
```

Reverse First K Element from Queue

Problem Statement - Given a queue and number K. Your task is to reverse it till K elements and then keep the remaining elements as it is.

For example –

Input – [1,2,3,4,5,6,7,8,9,10], k = 5
Output – [5,4,3,2,1,6,7,8,9,10]
Approach - We need to reverse the first k elements from the queue. To reverse, we will use Stack here. First we will dequeue front elements and push them to the stack. Then we will enqueue those elements again to the queue. This will add elements to the queue but at the rear side. Now, we will dequeue elements(size-k) from the front and enqueue them again to the same queue. This will give us our required queue.
Steps -

1) Create an empty stack
2) Dequeue k items from the queue and push them to the stack where k is given in the input
3) Enqueue k elements from stack to queue
4) Dequeue elements(size-k) from the queue and enqueue again to the front.

Reverse First K Element from Queue

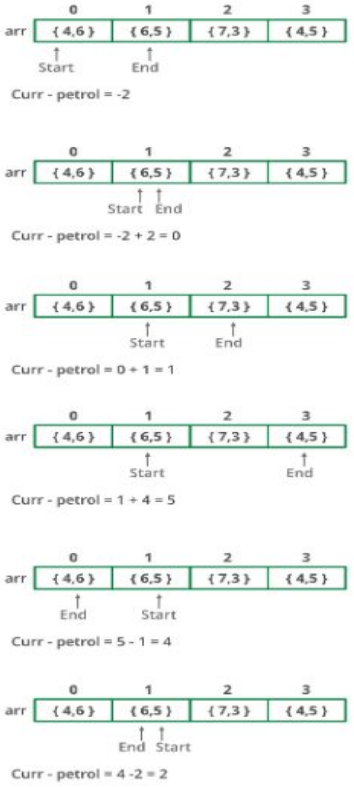
Code Link -> https://jsfiddle.net/xeh4b6Lv/

```
function reverseQueueFirstKElements(k) {
  if (queue.length < 0 ||
    k > queue.length)
    return;

  if (k <= 0)
    return;

  var stack = []


  for (i = 0; i < k; i++) {
    stack.push(queue[queue.length - 1]);
    queue.pop();
  }


  while (stack.length > 0) {
    queue.push(stack[stack.length - 1]);
    stack.pop();
  }


  for (i = 0; i < queue.length - k; i++) {
    queue.push(queue[queue.length - 1]);
    queue.pop();
  }
}

function print() {
  while (queue.length > 0) {
    console.log(queue[queue.length - 1]);
    queue.pop();
  }
}
```

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)



Circular Tour

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.

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

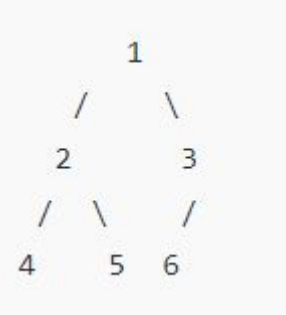
Code Link - https://jsfiddle.net/1xchrn59/

```
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}`);
```

Practice Questions

1) Given a binary tree. Your task is to find the height of it. Height is the number of nodes present in the longest path of the tree.

Input –



Output – 3

2) Given a binary tree. Your task is to print tree nodes in a zigzag manner.

Input -

Output – 1,3,2,4,5,6,7

Thank you