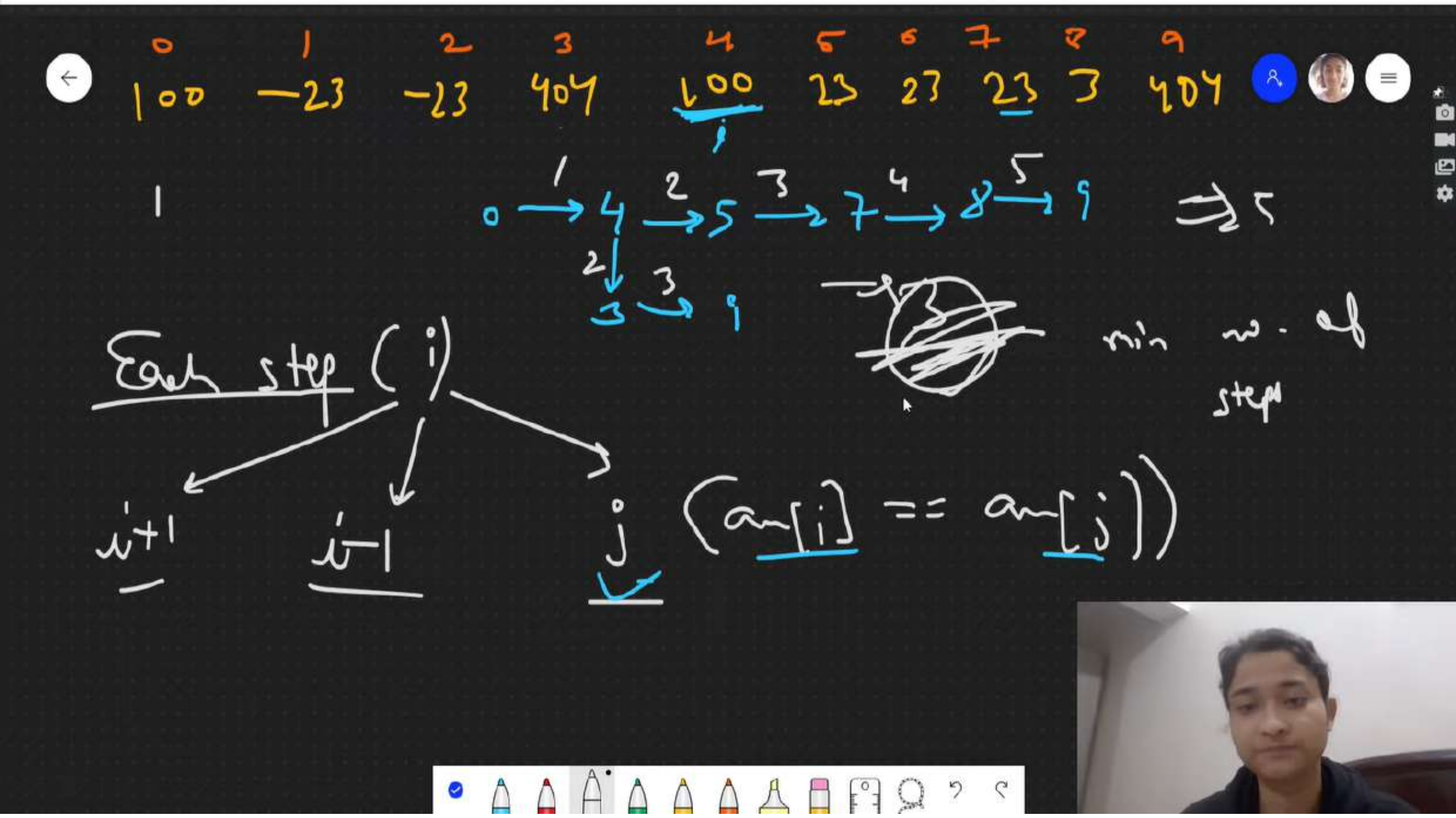


0	1	2	3	4	5	6	7	8	9
100	-23	-23	404	100	23	23	23	3	404

1

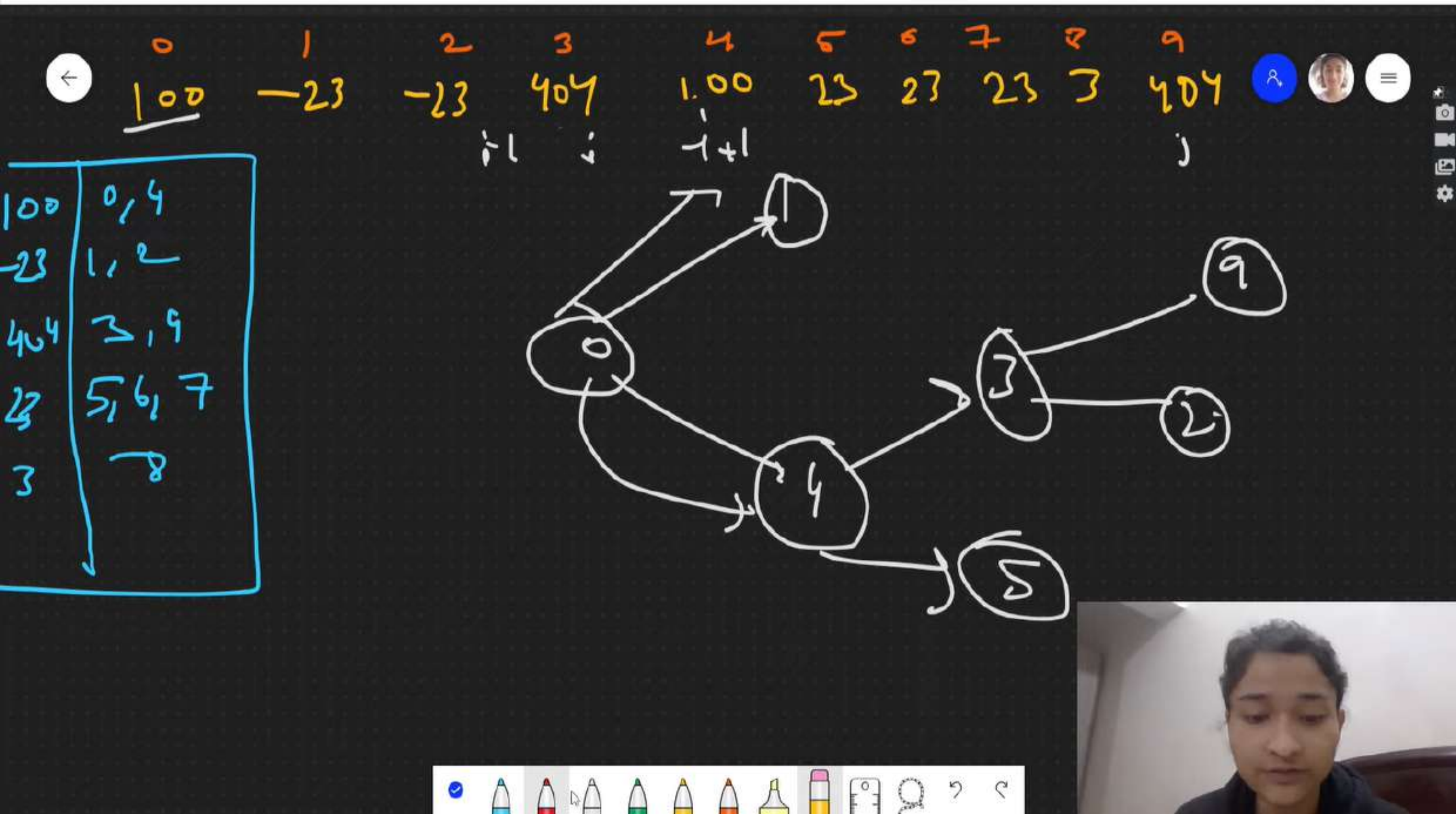
$0 \xrightarrow{1} 4 \xrightarrow{2} 5 \xrightarrow{3} 7 \xrightarrow{4} 8 \xrightarrow{5} 9 \Rightarrow 5$

$4 \xrightarrow{2} 3 \xrightarrow{3} 9$

min no. of steps

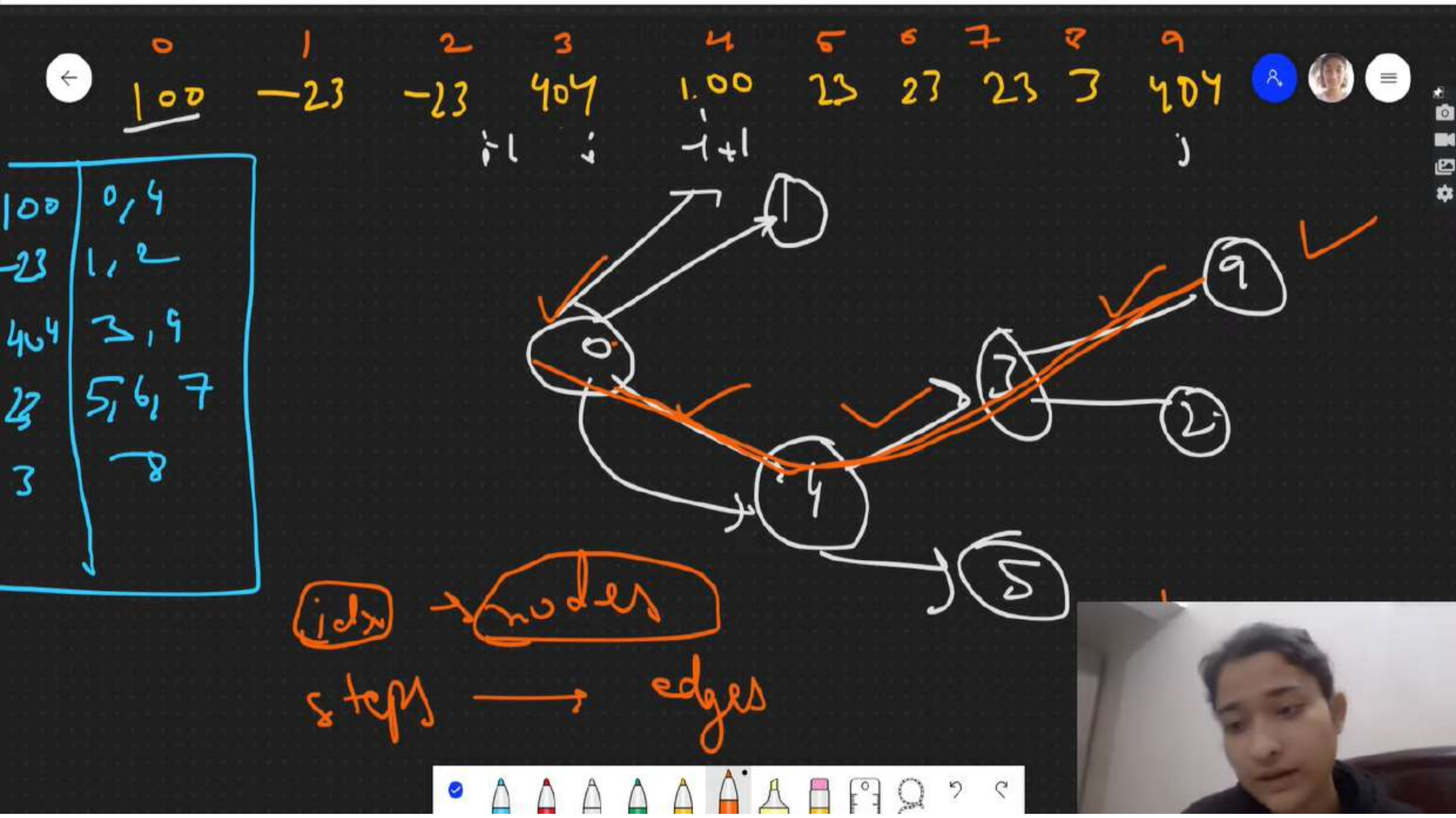
Each step (i)

i+1   i-1   j (arr[i] == arr[j])



0	1	2	3	4	5	6	7	8	9
100	-23	-23	404	1.00	23	23	23	3	404
		i-1	i	i+1					j

100	0, 4
-23	1, 2
404	3, 9
23	5, 6, 7
3	8

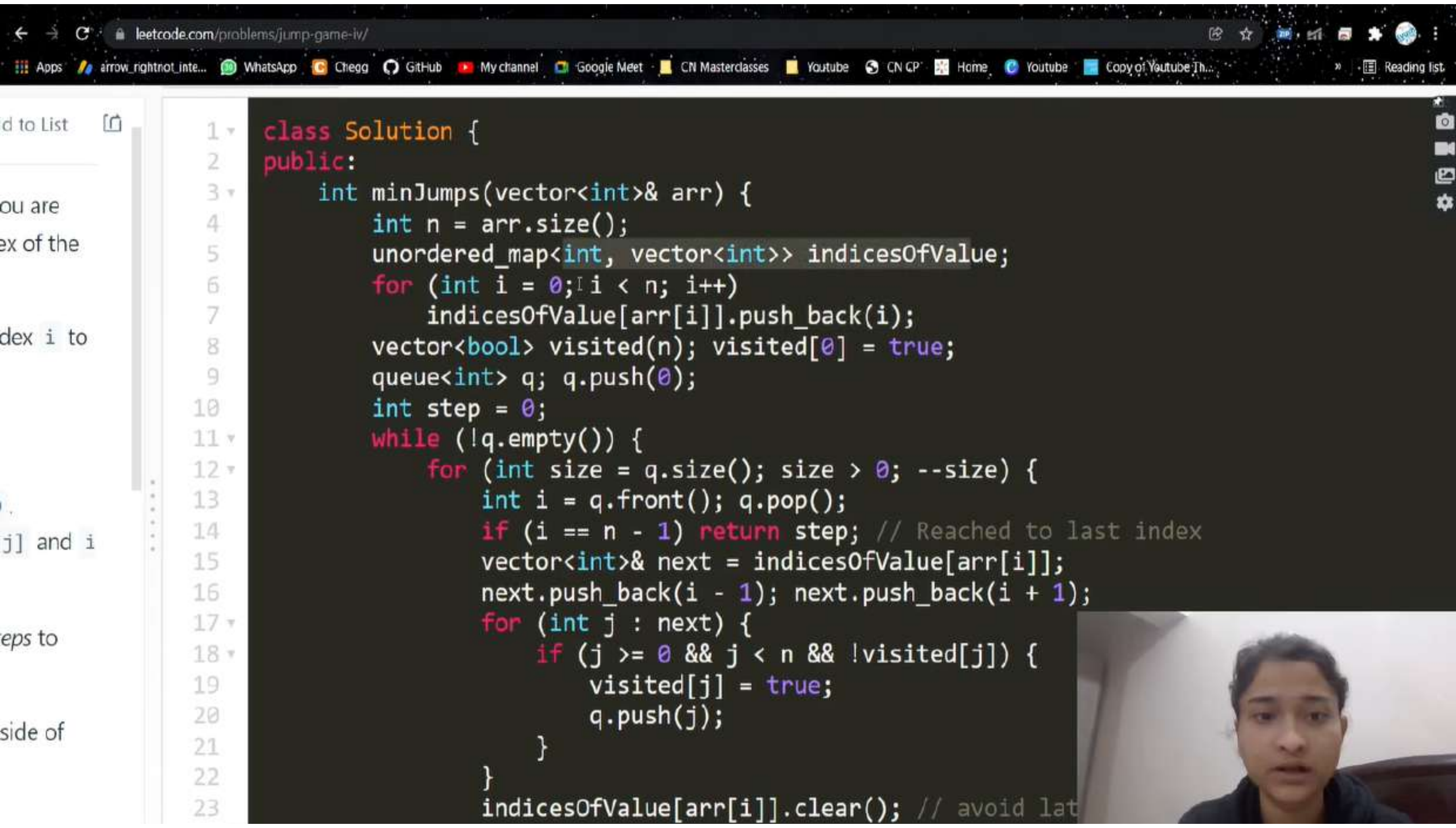


0	1	2	3	4	5	6	7	8	9
100	−23	−23	404	100	23	23	23	3	404
		i-1	i	i+1					j

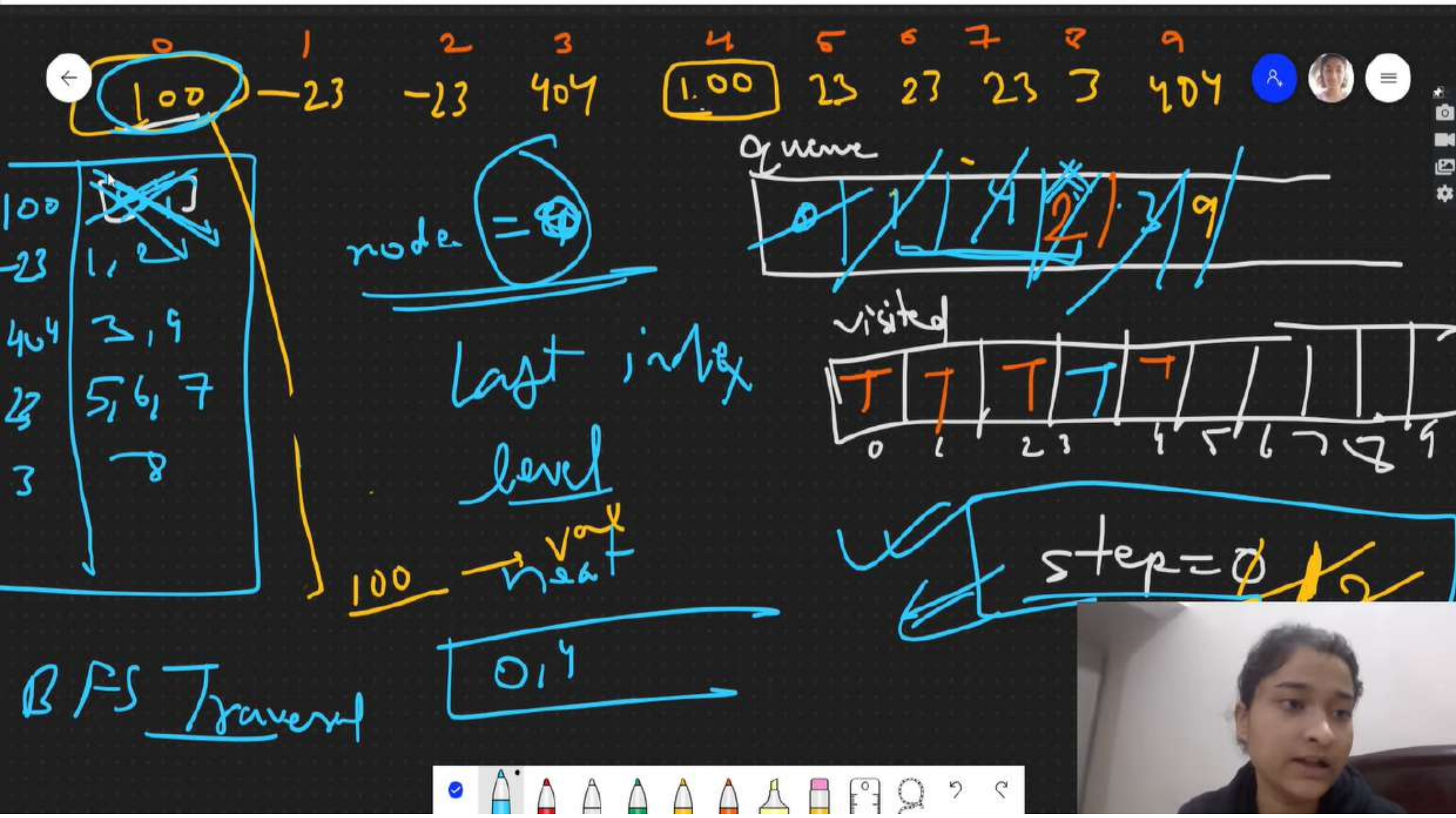
100	0, 4
-23	1, 2
404	3, 9
23	5, 6, 7
3	8

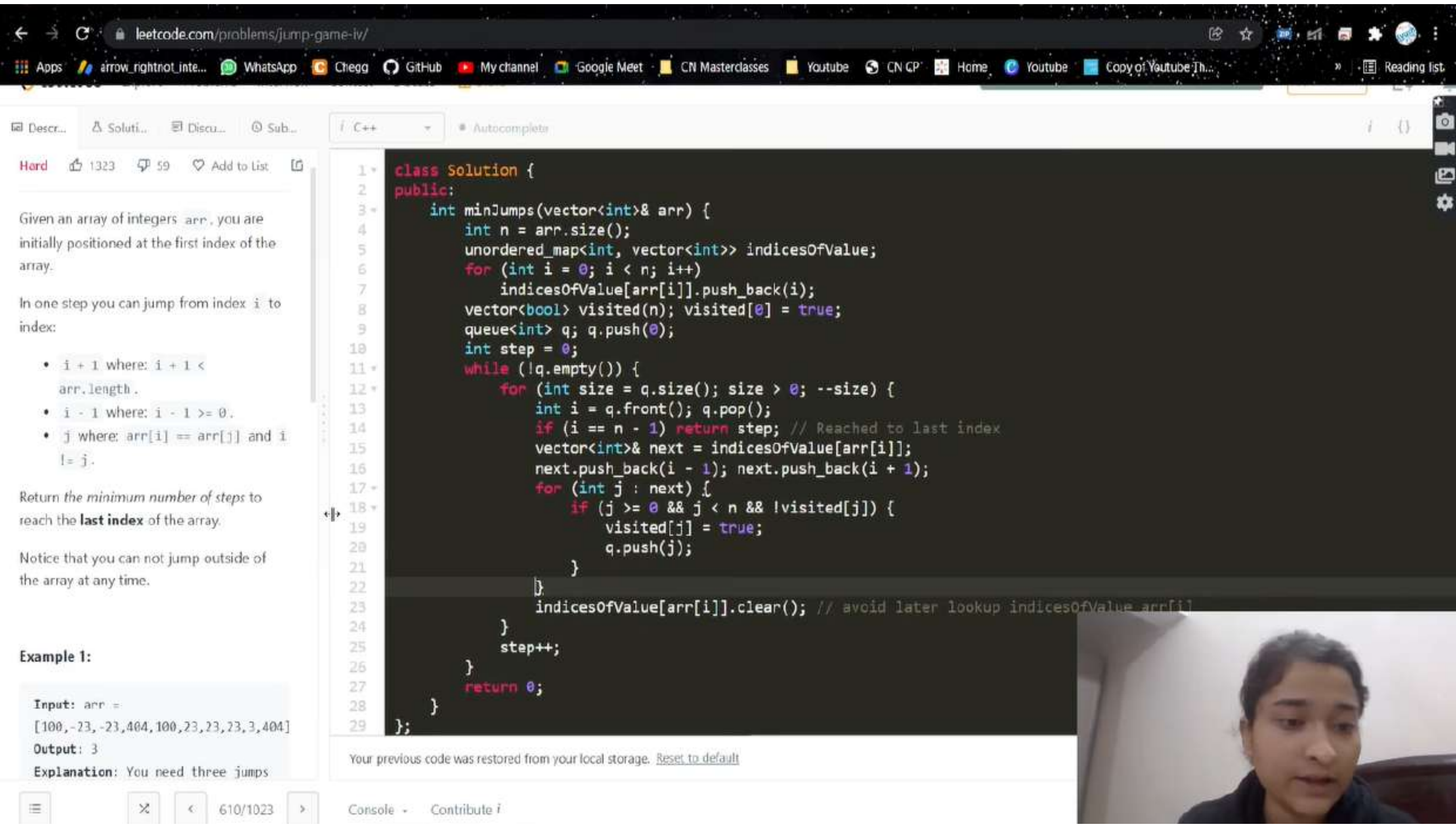
idx → nodes

steps → edges



```cpp
class Solution {
public:
    int minJumps(vector<int>& arr) {
        int n = arr.size();
        unordered_map<int, vector<int>> indicesOfValue;
        for (int i = 0; i < n; i++)
            indicesOfValue[arr[i]].push_back(i);
        vector<bool> visited(n); visited[0] = true;
        queue<int> q; q.push(0);
        int step = 0;
        while (!q.empty()) {
            for (int size = q.size(); size > 0; --size) {
                int i = q.front(); q.pop();
                if (i == n - 1) return step; // Reached to last index
                vector<int>& next = indicesOfValue[arr[i]];
                next.push_back(i - 1); next.push_back(i + 1);
                for (int j : next) {
                    if (j >= 0 && j < n && !visited[j]) {
                        visited[j] = true;
                        q.push(j);
                    }
                }
                indicesOfValue[arr[i]].clear(); // avoid lat
```


0 1 2 3 4 5 6 7 8 9
100 -23 -23 404 1.00 23 23 23 3 404
100 [0,4]
-23 1, 2
404 3, 9
23 5, 6, 7
3 8
node = 0
Last index
level
100 → next val
0, 4
BFS Traversal
Queue
0 1 4 2 1 3 9
visited
T T T T T
0 1 2 3 4 5 6 7 8 9
step = 0 1 2



Given an array of integers `arr`, you are initially positioned at the first index of the array.

In one step you can jump from index `i` to index:

- `i + 1` where: `i + 1 < arr.length`.
- `i - 1` where: `i - 1 >= 0`.
- `j` where: `arr[i] == arr[j]` and `i != j`.

Return *the minimum number of steps* to reach the **last index** of the array.

Notice that you can not jump outside of the array at any time.

Example 1:

```
Input: arr =
[100,-23,-23,404,100,23,23,23,3,404]
Output: 3
Explanation: You need three jumps
```

```cpp
class Solution {
public:
    int minJumps(vector<int>& arr) {
        int n = arr.size();
        unordered_map<int, vector<int>> indicesOfValue;
        for (int i = 0; i < n; i++)
            indicesOfValue[arr[i]].push_back(i);
        vector<bool> visited(n); visited[0] = true;
        queue<int> q; q.push(0);
        int step = 0;
        while (!q.empty()) {
            for (int size = q.size(); size > 0; --size) {
                int i = q.front(); q.pop();
                if (i == n - 1) return step; // Reached to last index
                vector<int>& next = indicesOfValue[arr[i]];
                next.push_back(i - 1); next.push_back(i + 1);
                for (int j : next) {
                    if (j >= 0 && j < n && !visited[j]) {
                        visited[j] = true;
                        q.push(j);
                    }
                }
                indicesOfValue[arr[i]].clear(); // avoid later lookup indicesOfValue arr[i]
            }
            step++;
        }
        return 0;
    }
};
```

Your previous code was restored from your local storage. Reset to default