MEDIUM

Minimum Multiplications to Reach End

Intuition

Given : Start, End and array of n elements

At each step start is multiplied with any number and we have to reach end

Eg.

Start = 3
End = 30

Arr[] = [2,5,7]

3*2 = 6
3*5 = 15
3*7 = 21….continuing we need to multiply again and again to reach 30 with minimum multiplications

Here 3*5*2 = 30 [one possible answer] ie. 2 multiplications

We have to mod with 100000 as per the question eg. 4095*65 = 266175%100000 = 66175… so we don't run to infinity

Note : Any number can be multiplied for any number of times

3

6,1 15,1 21,1

12,2 30,2 42,2 30,2 75,2 105,2 42,2 105,2 147,2
.
.
.

Mod will work for numbers only above 9999 so we can say that the points typically can range between 0-9999

Eg.

Distance :

0	1	2	3	4	5	6……15…9999	
			0			1	1

Queue :

[0,3]

[1,6]
[1,15]
[1,21]

[1,6]
[1,15]
[1,21]
[1,12]
[1,30]
[1,42]
.
.
.
.
[…,75] // return steps

We needed to figure out the mode here mainly

Queue will already store everything in increasing order so we don't need to use priority queue

Approach

- Check if the start is already equal to the end :
    - Return 0
- Create a queue to store [multiplied number, steps] as pair<int, int>
- Insert the start node with 0 steps into the queue
- Create a distance vector having mod number of elements all assigned infinity
- Mark the source distance as 0
- Traverse until the queue becomes empty :
    - Extract the first element of the queue
    - Pop the first element of queue
    - Traverse for the elements in the array :
        - Calculate number as number = original * ith number

        - Check if steps required to reach number + 1 are less than the distance to reach the number :
            - Update the distance in the distance vector
            - Check if the number is same as the required end :
                - Return the steps + 1
            - Push the updated distance with the element into the queue
    - Return -1 as we cannot reach the number in this case

Function Code

```
int minimumMultiplications(vector<int>& arr, int start, int end) {
        // if start and end are same then we need to return 0
        if(start==end)
        {
            return 0;
        }
        // Creating a queue to store a pair of multiplication and steps
        queue<pair<int, int>> q;
        // pushing the start value with 0 steps into the vector
        q.push({start,0});
        // creating a vector having all the elements of size mod in
assigned with infinity initially
        vector<int> distance(100000,1e9);
        // marking the start to have distance 0
        distance[start] = 0;
        // traversing until the queue becomes empty
        while(!q.empty())
        {
            // extracting the first node from the queue
            int node = q.front().first;
            int steps = q.front().second;
            // popping the node from the queue
            q.pop();
            // traversing for the adjacent elements
            for(int i:arr)
            {
                // finding the number after multiplication
                int num = (node*i)%100000;
                // check if the steps+1 are smaller than the current
distance to reach
                if(steps+1<distance[num])
                {
                    // update the distance
```

```
                    distance[num] = steps+1;
                    // check if the number we reached is already the end
                    if(num==end)
                    {
                        // return the number of steps to reach
                        return steps+1;
                    }
                    // push the element to the queue
                    q.push({num,steps+1});
                }

            }

        }
        // return -1 because in this case we cannot reach end
        return -1;
    }
```

Time Complexity

O(100000*N)