**Maximum Occurrence in a Given Range**

**Principle & Working of Code**

We are given an array of n numbers in increasing order. Our task is to find the maximum occurring number in a given range.

We can use the segment tree method to solve this problem. The logic for this problem is that the input array should be in increasing order such that the occurrences of each character would be placed in the frequency array in a sequential and sorted order. The segment tree can be created which would the store the maximum frequency of its respective range [i,j]. A frequency array would be required for this logic to perform.

**Case 1:** The value of the numbers at index i and j for a given range are same (arr[i]=arr[j]). This can be considered as the easiest method to solve this problem. Since, arr[i]=arr[j], the numbers between these two indices would be exactly the same as the array is in increasing order. Thus, the solution is the count of all numbers between the index i and j including them (j-i+1).

**Case 2:** If The value of the numbers at index i and j for the given range are different, i.e. arr[i] != arr[j]If arr[i]!=arr[j], then there definitely exists another index p where, arr[i]=arr[p] and arr[i]!=arr[p+1]. This may result in an occasion called partial overlap where some occurrences of a specific number would lie in the leftmost region of the entered range and the rest would lie just before the range starts. Calling the inbuilt function (Range Maximum Query) would obviously result in a wrong answer.

The same situation would happen in the rightmost part also where some occurrences would lie in the rightmost region of the range and the rest would lie after the range ends. Thus, for this case, inside the given range we have to take the frequency count of the leftmost same numbers upto an index i and the frequency count of the rightmost numbers upto an index j. Calling the RMQ function between these index and taking the maximum count of all three would result in the output.

**Algorithm**

1. Read an array n elements in increasing order.
2. Create a recursive function to get the maximum value in a given range of array indexes.
3. Create a node with starting and ending indexes.
4. If the segment of the given node is a part of given range then return the min of the segment.
5. Return minimum of elements in range from starting index to ending index.
6. Create a recursive that constructs segment tree for array.
7. If there is one element in array then store it in current node of segment tree and return.
8. If there are more than on elements, then recur the left and right subtrees and store the minimum of two values in this node.
9. Declare a frequency array.
10. Count frequencies of all array elements.
11. Create frequency array by replacing the number in array to the number of times it has appeared in the array.
12. Build a segment tree from the frequency array and declare maxOcc to store the answer.
13. If numbers are same at the starting and ending index of the query, maxOcc=(qe-qs+1).
14. If numbers are different, partial overlap case occurs.
15. At last, Take the maximum of all three and print the maximum occurring element.

**Program**

#include <bits/stdc++.h>

using namespace std;

int mid(int x, int y)

{

return x + (y - x) / 2;

}

int RMQUtil(int\* xt, int xx, int xy, int px, int py,

int index)

{

if (px <= xx && py >= xy)

return xt[index];

if (xy < px || xx > py)

return 0;

int middle = mid(xx, xy);

return max(RMQUtil(xt, xx, mid, px, py, 2 \* index + 1),RMQUtil(xt, mid + 1, xy, px, py, 2 \* index + 2));

}

int RMQ(int\* xt, int n, int px, int py)

{

if (px < 0 || py > n - 1 || px > py) {

printf("Invalid Input");

return -1;

}

return RMQUtil(xt, 0, n - 1, px, py, 0);

}

int constructSTUtil(int arr[], int xx, int xy, int\* xt,

int xi)

{

if (xx == xy)

{

xt[xi] = arr[xx];

return arr[xx];

}

int middle = mid(xx, xy);

xt[xi] = max(constructSTUtil(arr, xx, mid, xt, xi \* 2 + 1),constructSTUtil(arr, mid + 1, xy, xt, xi \* 2 + 2));

return xt[xi];

}

int\* constructST(int arr[], int n)

{

int a = (int)(ceil(log2(n)));

int maximum\_size = 2 \* (int)pow(2, a) - 1;

int\* xt = new int[maximum\_size];

constructSTUtil(arr, 0, n - 1, xt, 0);

return xt;

}

int maxoccurrence(int arr[], int n, int px, int py)

{

int count\_arr[n + 1];

unordered\_map<int, int> cnt;

for (int i = 0; i < n; i++)

cnt[arr[i]]++;

for (int i = 0; i < n; i++)

f=count\_arr[i] = cnt[arr[i]];

int\* xt = constructST(count\_arr, n);

int maximumans;

if (arr[px] == arr[py])

maximumans = (py - px + 1);

else

{

int leftmost = 0, rightmost =0;

while (px > 0 && px <= py && arr[px] == arr[px - 1])

{

px++;

leftmost ++;

}

while (py >= px && py < n - 1 && arr[py] == arr[py + 1])

{

py--;

righmost ++;

}

maxOcc = max({leftmost, righmost,RMQ(xt, n, px, py)});

}

return maximumans;

}

int main()

{

int arr[] = { -5, -5, 2, 2, 2, 2, 3, 7, 7, 7 };

int n = sizeof(arr) / sizeof(arr[0]);

int px = 0;

int py = 9;

cout << "Maximum Occurrence in range is = "

<< maxoccurrence (arr, n, px, py) << endl;

px = 4;

py = 9;

cout << "Maximum Occurrence in range is = "

<< maximumOccurrence(arr, n, px, py) << endl;

return 0;

}

**Output**

Maximum Occurrence in range is 4

Maximum Occurrence in range is 3

**~Anurag N Nair**