**Big Data Analytics**

**Experiment No. 07**

**Title:** Implementing DGIM algorithm using any Programming Language.

**Theory:**

Suppose we have a window of length N on a binary stream. We want at all times to be able to answer queries of the form “how many 1’s are there in the last k bits?” for any k≤ N. For this purpose, we use the DGIM algorithm.

The basic version of the algorithm uses O(log2 N) bits to represent a window of N bits, and allows us to estimate the number of 1’s in the window with an error of no more than 50%.

To begin, each bit of the stream has a timestamp, the position in which it arrives. The first bit has timestamp 1, the second has timestamp 2, and so on.

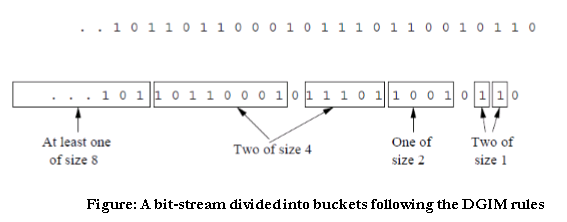
Since we only need to distinguish positions within the window of length N, we shall represent timestamps modulo N, so they can be represented by log2 N bits. If we also store the total number of bits ever seen in the stream (i.e., the most recent timestamp) modulo N, then we can determine from a timestamp modulo N where in the current window the bit with that timestamp is.

We divide the window into buckets, 5 consisting of:

1. The timestamp of its right (most recent) end.
2. The number of 1’s in the bucket. This number must be a power of 2, and we refer to the number of 1’s as the size of the bucket.

To represent a bucket, we need log2 N bits to represent the timestamp (modulo N) of its right end. To represent the number of 1’s we only need log2 log2 N bits. The reason is that we know this number i is a power of 2, say 2j , so we can represent i by coding j in binary. Since j is at most log2 N, it requires log2 log2 N bits. Thus, O(logN) bits suffice to represent a bucket. There are six rules that must be followed when representing a stream by buckets.

1. The right end of a bucket is always a position with a 1.
2. Every position with a 1 is in some bucket.
3. No position is in more than one bucket.
4. There are one or two buckets of any given size, up to some maximum size.
5. All sizes must be a power of 2.
6. Buckets cannot decrease in size as we move to the left (back in time).



**Program:**

ip = list(map(int, input(“Enter Elements “).split()))

bucket\_list = []

bucket\_size\_count = {}

def checker():

for ct in bucket\_size\_count.keys():

if(bucket\_size\_count[ct]>2):

s2, e2, size2 = bucket\_list.pop(-2)

s1, e1, size1 = bucket\_list.pop(-2)

bucket\_list.insert(-1, (s1,e2,size1\*2))

bucket\_size\_count[ct]-=2

start\_index = 0

end\_index = 0

pair = 0

for i in range(len(ip)):

bit = ip[i]

if(bit == 1):

if(pair == 1):

end\_index = i

pair=0

bucket\_list.append((start\_index, end\_index, 2))

if 2 in bucket\_size\_count:

bucket\_size\_count[2]+=1

else:

bucket\_size\_count[2]=1

checker()

else:

start\_index = i

pair=1

print("Bucket indexs are",bucket\_list)

starts = []

ends = []

for s,e,size in bucket\_list:

starts.append(s)

ends.append(e)

print("Buckets are",end=" ")

for i in range(len(ip)):

bit = ip[i]

if(i in starts):

print(" ",bit,end=" ")

elif(i in ends):

print(bit,end=" ")

else:

print(bit,end=" ")

k = int(input("\nEnter k "))

length = len(ip)

bound1 = length-1-k

bound2 = length-1

ones\_count=0

for s,e,size in bucket\_list[::-1]:

if(s<bound1 and e<bound1):

break

elif(s<=bound1 and e>=bound1):

ones\_count+=int(size/2)

break

elif(s>=bound1 and e>=bound1):

ones\_count+=size

print("Number of ones in last",k,"bits is",ones\_count)

**Output:**

Enter Elements 1 0 0 1 0 1 0 1 1 0 0 0 1 0 1 1

Buckets are 1 0 0 1 0 1 0 1 1 0 0 0 1 0 1 1

Enter k 5

Number of ones in last 5 bits is 3

**Conclusion:** We studied and implemented DGIM algorithm in Python.