Subarrays Intro 1

Sunday, 19 June 2022 10:06

Rotate array R=L { Rotation Frame }

R L

Ex: arr[7] = [0 1 2 3 4 5 6] = [-2 1 7 8 9]

Rotate 1 : [8 7 -2 1 9 6]

Rotate 2 : [9 8 7 -2 1 9 6]

Rotate 3 : [6 9 8 7 -2 1 9]

left 3 ↔ Remaining 4 to 7 ↔

Ex: arr[7] = [3 -2 8 0 10 5 -1]

Rotate by 4 : 10 3 -2 8 0 5 -1

Ex: arr[13] = [0 1 2 3 4 5 6 7 8 9 10 11 12] = [8 9 10 11 12 0 1 2 3 4 5 6 7]

R=6 → [8 9 10 11 12 0 1 2 3 4 5 6 7]

N-R

R elements

Reverse entire → [12 11 10 9 8 6 5 4 3 2 1 0 7]

Reverse Subarray — Reverse the subarray

[8 9 10 11 12 0 1 2 3 4 5 6 7]

Ex: Rotate array R times

Steps: Reverse the array : Iterations : N/2

Steps: Reverse first R elements & Reverse Subarray [0, R-1] : R/2

Steps: Reverse Remaining N-R elements & Reverse Subarray [R, N-1] : N-R/2

Total : N , SC : O(1)

Ex: Rotate array R times

Step 1: Reverse the array : Iterations : N/2

Step 2: Reverse first R elements & Reverse Subarray [0, R-1] : R/2

Step 3: Reverse Remaining N-R elements & Reverse Subarray [R, N-1] : N-R/2

Total : N , SC : O(1)

Ex: arr[7] = [a0 a1 a2 a3 a4 a5 a6]

Rotate by 0 : a0 a1 a2 a3 a4 a5 a6

Rotate by 1 : a1 a0 a3 a2 a5 a4 a6

Rotate by 2 : a2 a3 a0 a1 a6 a5 a4

Rotate by 3 : a3 a2 a1 a0 a4 a6 a5

Rotate by 4 : a4 a5 a6 a0 a1 a2 a3

Rotate by 5 : a5 a4 a6 a1 a0 a3 a2

Rotate by 6 : a6 a5 a4 a2 a3 a1 a0

Rotate

0 → 0 → 0 → 0

1 → 1 → 1 → 1

2 → 2 → 2 → 2

3 → 3 → 3 → 3

4 → 4 → 4 → 4

5 → 5 → 5 → 5

6 → 6 → 6 → 6

0 = 12 % 7 = 5

1 = 13 % 7 = 6

2 = 14 % 7 = 0

3 = 15 % 7 = 1

4 = 16 % 7 = 2

5 = 17 % 7 = 3

6 = 18 % 7 = 4

(R % L) = k → length of array

Given N, Rotate by k times

R = k % N

2.b) Given an array, Print all subarray sums (Contiguous) N ≤ 10^5

arr[] : 3 -2 4 -1 2 6 — contains part from array [0, 5]

Single ↔

Window ↔

[3] → 6

Subarrays

Index = 0

[0] : 3

[0, 1] : 1

[0, 2] : 5

[0, 3] : 4

[0, 4] : 6

[0, 5] : 12

Index = 1

[1] : -2

[1, 2] : 2

[1, 3] : 1

[1, 4] : 3

[1, 5] : 9

Index = 2

[2] : 4

[2, 3] : 3

[2, 4] : 5

[2, 5] : 11

Index = 3

[3] : -1

[3, 4] : 1

[3, 5] : 7

Index = 4

[4] : 2

[4, 5] : 8

Index = 5

[5] : 6

Logic : For every subarray calculate Sum & print it.

T = 0; T < N; T++ {

J = T; J < N; J++ {

S = S + arr[J]

print(S)

}

}

TC : O(N^2) → TLE

Given Sum & All [2, 5] → 2

Prefix Sum

S = Sum of all Indices = [T, J]

Sum[T, J] = Sum[0, J] - arr[T-1]

Sum[T, J] = S - arr[T-1]

T = 0; T < N; T++ {

S = 0

J = T; J < N; J++ {

S = S + arr[J]

print(S)

}

}

TC : O(N^2)

SC : O(1)

Trace:

arr[] = 3 -2 4 1 6

0 1 2 3 4

T = 0

S = 0

J = 0, J < 5, J++ {

S = S + arr[0] = 0 + 3

S = 3 print(S)

J = 1, S = S + arr[1] = 3 + -2

S = 1 print(S)

J = 2, S = S + arr[2] = 1 + 4

S = 5 print(S)

J = 3, S = S + arr[3] = 5 + 1

S = 6 print(S)

J = 4, S = S + arr[4] = 6 + 6

S = 12 print(S)

T = 1

S = 0

J = 1, J < 5, J++ {

J = 1, S = S + arr[1]

S = -2 print(S)

J = 2, S = S + arr[2]

S = 2 print(S)

J = 3, S = S + arr[3]

S = 1 print(S)

J = 4, S = S + arr[4]

S = 7 print(S)

3.a) Print Max and Min Index of all subarrays [a, b] = b - a ≤ 1

arr[] : 0 1 2 3 4 5 6 7 8 9 10 11

3 4 2 11 6 7 8 9 3 2 -1 4

R = 6

Output

1st Subarray : 0 5

2nd Subarray : 1 6

3rd Subar. : 2 7

4th : 3 8

5th Subr. : 4 9

⋮

last Subar. : 6 11

N elements & print all Subr. = k

1st Subarray : 0 1

2nd subarray : 1 2

3rd Subar. : 2 3

4th Subar. : 3 4

Last Subarray : N-k, N-1

k-1

T = 0; T ≤ N-k; T++ {

J = T, J < T+k; J++ {

print(J, J+k-1)

}

}

TC : O(N·k)

[ ] [ ] → k

T = T + 1 = k

T = T + 1 = k

Non-of elements = k

3.b) Find Max subarray Sum with length k

arr[] : -3 4 -2 5 3 -2 8 2 -1 4

k = 5

Output : 16

Idea : Sum of all subarrays of length k & get max of them.

MAX = INT_MIN

T = 0; T ≤ N-k; T++ {

J = T; J < T+k; J++ {

S = S + arr[J]

}

K = T, K < J, K++ {

S = S - arr[K]

}

// As have a subarray sum [T, J]

if (max < S) { max = S }

}

TC : O(N·k)

SC : O(1)

Less optimal Method

Step 2

arr[] : -3 4 -2 5 3 -2 8 2 -1 4

(k-1)

Sum

Sum

[0, 5] → 5

[1, 6] = 5 - arr[0] + arr[6]

= 5 - (-3) + 8 = 16

[2, 7] = 16 - arr[1] + arr[7]

16 - 4 + 2 = 14

[3, 8] = 14 - arr[2] + arr[8]

14 - (-2) + -1 = 15

Initialize N Array elements, subarray k increase & get sum

1st Subarray : 0 1 2 3 4

2nd Subarray : 1 2 3 4 5

3rd Subarray : 2 3 4 5 6

4th Subarray : 3 4 5 6 7

last Subar. : N-k, N-1

S = S - arr[T-1] + arr[T+k-1]

Step 1 : S = 0

T = 0; T < k; T++ {

S = S + arr[T]

}

max = S // S is first subarray sum

T = 1 ; T < N-k; T++ {

J = T+k-1

// S = S - arr[T-1] + arr[J]

S = S - arr[T-1]

S = S + arr[J]

if (max < S) { max = S }

}

TC : O(N)

SC : O(1)

4.a) Find Sum of All Subarray Sums

arr[] : 3 -2 4 1 2 6

Subarrays

Index = 0

[0] : 3

[0, 1] : 1

[0, 2] : 5

[0, 3] : 4

[0, 4] : 6

[0, 5] : 12

Index = 1

[1] : -2

[1, 2] : 2

[1, 3] : 1

[1, 4] : 3

[1, 5] : 9

Index = 2

[2] : 4

[2, 3] : 3

[2, 4] : 5

[2, 5] : 11

Index = 3

[3] : -1

[3, 4] : 1

[3, 5] : 7

Index = 4

[4] : 2

[4, 5] : 8

Index = 5

[5] : 6

Sum of all subarrays : ?

arr[] = 0;

T = 0; T < N; T++ {

S = 0;

J = T; J < N; J++ {

S = S + arr[J]

Sum = Sum + S

}

}

return Sum;

TC : O(N^2)

4.b) Find Sum of all Subarray Sums

arr[] : 3 -2 4 1 2 6

1st Subarray : 0 1 2 3 4 5

2nd Subarray : 1 2 3 4 5

3rd Subarray : 2 3 4 5

4th Subarray : 3 4 5

5th Subarray : 4 5

last Subar. : 5

Sum = S + arr[T] + arr[T+1] + arr[T+2] + arr[T+3] + arr[T+4] + arr[T+5]

Step 1 : S = 0

T = 0; T < k; T++ {

S = S + arr[T]

}

Step 2 : S = 0

T = 0; T < N; T++ {

S = S - arr[T-1] + arr[T+k-1]

Sum = Sum + S

}

return Sum

TC : O(N)

SC : O(1)

Ex: arr[] : 3 -2 4 -1 2 6

0 1 2 3 4 5

left -2 → 5 = 10

Knowledge: [a0 a1 a2 a3] [a4 a5] [a6] — [0, N-1] = [N-y-1, N-1]

Right = [1]

Right = N-1

No. of subarrays in which arr[T] is present = (T+1) * (N-T)

Ex: arr[2] = 4, T+1 = 3, N-T = 4

We are adding contribution of every element

S = 0;

T = 0; T < N; T++ {

left = T+1, right = N-T

total = left * right

S = S + total * arr[T]

}

return S

Total = left * right → Then many times it will come in subarrays

TC : O(N)

SC : O(1)

Left Right

T = 0 : 1 2 3 4 5 6 7

left : 1 2 3 4 5 6 7

right : 7 6 5 4 3 2 1

Sum : 1*7 + 2*6 + 3*5 + 4*4 + 5*3 + 6*2 + 7*1

= 7 + 12 + 15 + 16 + 15 + 12 + 7 = 84

→ Sum = 84

Output

arr[] = [1, 2, 3, 4, 5]

to

Result = [1, 1+2, 1+2+3, 1+2+3+4, 1+2+3+4+5]

arr[] : 0 1 2 3 4 5 6 7 8

-3 4 -2 7 8 1 6 2 9

[0, 0] :

[1, 8] :

[2, 8] :

[