Tricky Arrays

Contributor: Admin

Contents

1) Regards
2) Why Arrays?
3) Problems
   a) Sum entire array
   b) Reverse entire array
   c) Subarray?
   d) Reverse Subarray
   e) Rotate entire array
   f) Max subarray sum
   g) Subsequence? Subset
   h) Tic Array problem
   i) Dynamic Array Implementation?
4) Doubts

1) Regards

Hello!

Yesterday: 1 one person to B

A → B, B → C, C → B

[B, 100] [Connection] [A] [B Connection] [fire]

Now a, b, y, d, y, f = Array. Multiple notes + done. Type + clear.

Ex: a[2]

0 1 2 3 4 5 6 7 8

10 20 30 40 50 60 70 80 90

to delete value: a[1] = a[2], a[2] = a[3]

Now a, b, c, d ... → 2, 3, 6

Ex: a[2]

0 1 2 3 4 5 6 7 8

10 20 30 40 50 60 70 80 90

Now Time

0 1 2 3 4 5 6 7

a[2] = 3 5 7 11 13 19 20 0

Save the entire observed array?

Solution: 1) [0, 2] saved

2) Use new position → [log n] → Not Above

Ex: a = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

Time: O(n)

Space: O(1)

2) Why Arrays?

Ex: 0 1 2 3 4 5 6 7 8

3 5 7 11 13 19 20

Value Can: (N-1) : O(N-1)

Space: 1 → Dump

3 5 7 11 13 19 20

Dump

3 5 7 11 13 19 20

Dump

3 5 7 11 13 19 20

Add Group Break

Not: a = 10

Not: b = 20

c = a, a = b, b = c

Swapping with temp variables

N: Elements are Stored

D: Size are 4 Elements

N: Array elements

0, 1, 2, 3, 4 → N-1 → Swapping array elements

N-1, N-2, N-3, N-4, N-5, N-6

for (i = N-1; i > 0; i--) {

if (arr[i] < arr[i-1]) {

swap arr[i] + arr[i-1]

break

}

}

0 1 2 3 4

arr[2] 3 5 7 11 13

i 3 5 = arr[2] = 5 Dump

2 1 0

3 5 7 11 13

arr[2] = arr[1]

arr[1] = arr[0]

Array did + Sorted error

Approach:

Ex: 0 1 2 3 4 5 6 7 8 9

arr[i] 3 5 7 11 13 19 20 10

Approach: 1) Use arr[] linear Find → 2) copy to tp[i] i = 0 → 7 → O(n)

0 1 2 3 4 5 6 7 8 9

10 20 30 40 50 60 70 80 90

2) Copy tp[i] → arr[i]

Time: O(n), Space: O(n)

Issues: Extra memory is like the problem.

Subarrays: Continuous part of an array is called Subarray

→ Is single element Subarray? Yes

→ Entire Array Subarray? Yes

arr[i] = 0 1 2 3 4 5 6

3 5 7 11 13 19 20

Subarray

(Not continuous)

+ Subarray

Number: 1, 2, 3

arr = 3 0 2 1 5 6

How many Subarray are there?

arr[i] = 0 1 2 3 4 5 5 = 5 + 1

[3] [5] [7] [11] [13] [19] [20]

[3, 5] [5, 7] [7, 11] [11, 13] [13, 19] [19, 20]

[3, 5, 7] [5, 7, 11] [7, 11, 13] [11, 13, 19] [13, 19, 20]

[3, 5, 7, 11] [5, 7, 11, 13] [7, 11, 13, 19] [11, 13, 19, 20]

[3, 5, 7, 11, 13] [5, 7, 11, 13, 19] [7, 11, 13, 19, 20]

[3, 5, 7, 11, 13, 19] [5, 7, 11, 13, 19, 20]

[3, 5, 7, 11, 13, 19, 20]

5 4 3 2 1

N Elements = (N)(N+1) / 2

Brute force to Example

Ex: a = 10, b = 20

a = a^b

b = a^b

a = a^b

a b

b a

Subarrays

Ex: 0 1 2 3 4 5 6 7 8 9

arr[i] = 3 2 6 7 2 7 6 5 7 10

5 = 2 + 3

ct = 5

Reverse Total Subarrays

Subarrays

Reversing Subarrays

Ex: 0 1 2 3 4 5 6 7 8 9

arr[i] = 3 2 6 7 2 7 6 5 7 10

Reverse: Array [2, 5] → True

Ex: Reverse and Stop or Done?

3 2 6 7 2 7 6 5 7 10

10 7 5 6 7 2 7 6 2 3

Rotate: 1 Time

Ex: 0 1 2 3 4 5 6 7 8 9

arr[i] = 3 2 6 7 2 7 6 5 7 10

Rotate: 1 Time

Swap arr Elements from a to n-1, to right

a, arr[N-1]

for (i = N-1; i > 0; i--) {

arr[i] = arr[i-1]

}

arr[0] = x

→ Single Rotate: O(n)

Array have 2 arrays + [3 ints, Size 2 Array]

Ex: 0 1 2 3 4 5 6 7 8 9 10

arr[i] = 3 2 6 7 2 7 6 5 7 10

Rotate: 1 Time

Swap arr Elements from a to n-1, to right

a, arr[N-1]

for (i = N-1; i > 0; i--) {

arr[i] = arr[i-1]

}

arr[0] = x

→ Single Rotate: O(n)

Array have 2 arrays + [3 ints, Size 2 Array]

Ex: 0 1 2 3 4 5 6 7 8 9 10

arr[i] = 3 2 6 7 2 7 6 5 7 10

Rotate: 1 Time

Swap arr Elements from a to n-1, to right

a, arr[N-1]

for (i = N-1; i > 0; i--) {

arr[i] = arr[i-1]

}

arr[0] = x

→ Single Rotate: O(n)

Array have 2 arrays + [3 ints, Size 2 Array]

Solution Steps:

1) Reverse Entire Array → n/2

2) Reverse Subarray [0, k-1] → k/2

3) Reverse Subarray [k, N-1] → (N-k)/2

N Elements

Time: O(n)

arr[i] = 3 4 2 7 8

k = 2 → 3 4 2 7 8

Back to Subarray Space: O(n)

→ Find arr[i+2] { N Elements } Static Arrays?

Dynamic Arrays?

C++: Vector

Java: ArrayList

Python: List

Ruby: Array[]

C#: List

Golang: Slice

Step Size: 2(n), Step Size = 10

for (i = 0; i < N; i++) {

arr[i] = i + 10

}

Adding: To add any array element → O(n)

Ex: From N dynamic Arrays [Implement + Write]

→ To add any element → O(1) time

→ To pop any element → O(1) time

Time: O(n)

Space: O(n) → Not Optimal

Todo: In your language + check

Please read how to use dynamic array

i = 1; i < N; i = i + 2

1, 3, 5, 7, 9, 11, 13 / 2

N/2

i = 1; i < (2n); i = i + 2

i = 1, 3, 5, 7, 9, 11, 13, 15, 17

(2n)/2

for (int i = 0; i < N) {

for (int j = 0; j < N; j++) {

}

}

for (int i = 0; i < N) {

for (j = 0; j < N; j++) {

}

}

→ N

→ N

+N

O(N)

i	
i=0	—
i=1	[0, N]
i=2	[0, N]
i=3	[0, N]
i=4	[0, N]
i=N-1	[0, N]

1

2

3

4

N-1

N²

i = 0

for (int i = 0; i < N) {

j = 0

while (j < i) {

j = j + 1

}

}

i	
i=0	—
i=1	[0, 0]
i=2	[0, 1]
i=3	[0, 2]
i=4	[0, 3]
i=N-1	[0, N-1]

1

2

3

4

N-1

(N-1)(N-0), O(N²)

0 1 2 3 4 5 6 7 8

arr[i] = 3 4 8 9 10 12 14 16