Prefix Sums

Ieuan David Vinluan

July 2024

Introduction

Let us first start with a problem. Given an array A of integers of length n:

1	4	3	5	2

Introduction

Let us first start with a problem. Given an array A of integers of length n:

1	4	3	5	2

For q queries, find the sum S of the elements from a starting index l to an ending index r.

Introduction

Let us first start with a problem. Given an array A of integers of length n:

1	4	3	5	2

For q queries, find the sum S of the elements from a starting index l to an ending index r. For our example above:

- if $(l, r) = (0, 4)$, $S = 1 + 4 + 3 + 5 + 2 = 15$
- if $(l, r) = (1, 2)$, $S = 4 + 3 = 7$
- if $(l, r) = (2, 2)$, $S = 3$

Introduction

Coding this solution is pretty straightforward. For each query, just go through each element within the range and track the sum!

Introduction

Coding this solution is pretty straightforward. For each query, just go through each element within the range and track the sum!

```
for (int i = 0; i < q; i++) {
    int ans = 0;
    cin >> l >> r;
    for (int j = l; j <= r; j++) {
        ans += A[j];
    }
    cout << ans << endl;
}
```

Introduction

Let's analyze the time complexity of this algorithm.

Introduction

Let's analyze the time complexity of this algorithm.

- Inner loop: $O(n)$

Introduction

Let's analyze the time complexity of this algorithm.

- Inner loop: $O(n)$
- Outer loop: $O(q)$

Let's analyze the time complexity of this algorithm.

- Inner loop: $O(n)$
- Outer loop: $O(q)$
- Thus, the entire solution has a time complexity of $O(nq)$

Let's analyze the time complexity of this algorithm.

- Inner loop: $O(n)$
- Outer loop: $O(q)$
- Thus, the entire solution has a time complexity of $O(nq)$

Is it possible to do better?

Insights

Let us go back to our example earlier. Let $S_{x,\,y}$ denote S for $(l, r) = (x, y)$. Consider the case where $(l, r) = (2, 3)$:

Insights

Let us go back to our example earlier. Let $S_{x,\,y}$ denote S for $(l, r) = (x, y)$. Consider the case where $(l, r) = (2, 3)$:

1	4	3	5	2

Insights

Let us go back to our example earlier. Let $S_{x,\,y}$ denote S for $(l, r) = (x, y)$. Consider the case where $(l, r) = (2, 3)$:

1	4	3	5	2

In this case, $S = A_2 + A_3$. Observe that $S_{2,\,3}$ can be calculated by first computing the sum $S_{0,\,3}$ then subtracting the sum $S_{0,\,1}$:

Insights

Let us go back to our example earlier. Let $S_{x,\,y}$ denote S for $(l, r) = (x, y)$. Consider the case where $(l, r) = (2, 3)$:

1	4	3	5	2

In this case, $S = A_2 + A_3$. Observe that $S_{2,\,3}$ can be calculated by first computing the sum $S_{0,\,3}$ then subtracting the sum $S_{0,\,1}$:

1	4	3	5	2

1	4	3	5	2

Insights

We can see this algebraically, too.

Insights

We can see this algebraically, too.

$$\begin{aligned} S_{0,\,3} - S_{0,\,1} &= (A_0 + A_1 + A_2 + A_3) - (A_0 + A_1) \\ &= A_2 + A_3 \\ &= S_{2,\,3} \end{aligned}$$

Insights

We can see this algebraically, too.

$$\begin{aligned} S_{0,\,3} - S_{0,\,1} &= (A_0 + A_1 + A_2 + A_3) - (A_0 + A_1) \\ &= A_2 + A_3 \\ &= S_{2,\,3} \end{aligned}$$

This can be generalized: $S_{x,\,y} = S_{0,\,y} - S_{0,\,x-1}$

So What?

The good thing is that our array remains constant, meaning that the sum from 0 to some index i will always be the same.

So What?

The good thing is that our array remains constant, meaning that the sum from 0 to some index i will always be the same.

Then, our new plan of attack would be to precompute all $S_{0,i}$ for all $0 \leq i \leq n-1$, which, as we have seen, will let us find the sum of values from any index l to another index r.

Our New Solution

Our first step would be to create our list `pre` of all $S_{0,\,i}$ for all indices i such that $0 \le i \le n-1$. We can use the fact that $S_{0,\,i} = S_{0,\,i-1} + A_{i-1}$.

Our New Solution

Our first step would be to create our list pre of all $S_{0,\,i}$ for all indices i such that $0 \leq i \leq n-1$. We can use the fact that $S_{0,\,i} = S_{0,\,i-1} + A_{i-1}$.

```
vector<int> pre(n);
pre[0] = A[0];
for (int i = 1; i < n; i++) {
    pre[i] = pre[i - 1] + A[i];
}
```

Our New Solution

Then, for each query of l and r, simply compute $S_{0,r} - S_{0,l-1}$. Note that you have to handle the edge case when $l = 0$.

Our New Solution

Then, for each query of l and r, simply compute $S_{0,r} - S_{0,l-1}$. Note that you have to handle the edge case when $l = 0$.

```
for (int i = 0; i < q; i++) {
    cin >> l >> r;
    if (l > 0) {
        cout << pre[r] - pre[l - 1] << endl;
    } else {
        cout << pre[r] << endl; // why?
    }
}
```

Our New Solution

Then, for each query of l and r, simply compute $S_{0,r} - S_{0,l-1}$. Note that you have to handle the edge case when $l = 0$.

```
for (int i = 0; i < q; i++) {
    cin >> l >> r;
    if (l > 0) {
        cout << pre[r] - pre[l - 1] << endl;
    } else {
        cout << pre[r] << endl; // why?
    }
}
```

Alternatively, you could shift all the values of pre one index to the right and define pre[0] to be 0. How will this change our code?

Analysis

Now, let us analyze our new solution.

Analysis

Now, let us analyze our new solution.

- Precomputation: $O(n)$

Analysis

Now, let us analyze our new solution.

- Precomputation: $O(n)$
- Handling q queries: $O(q)$

Analysis

Now, let us analyze our new solution.

- Precomputation: $O(n)$
- Handling q queries: $O(q)$
- Thus, the entire solution has a time complexity of $O(n + q)$, which is better for large values of n and q!

Other Notes

Other Notes

- Prefix Arrays can be applied to other binary operations, such as taking the GCD or LCM, addition, subtraction, and so on.

Other Notes

- Prefix Arrays can be applied to other binary operations, such as taking the GCD or LCM, addition, subtraction, and so on.
- Make sure that the array is static (i.e., constant after each query); if the elements of the array change, our precomputation is useless.

Homework :3c

Check the Reboot website! As always, you can ask for help in the Reboot Discord Server if you're stuck.