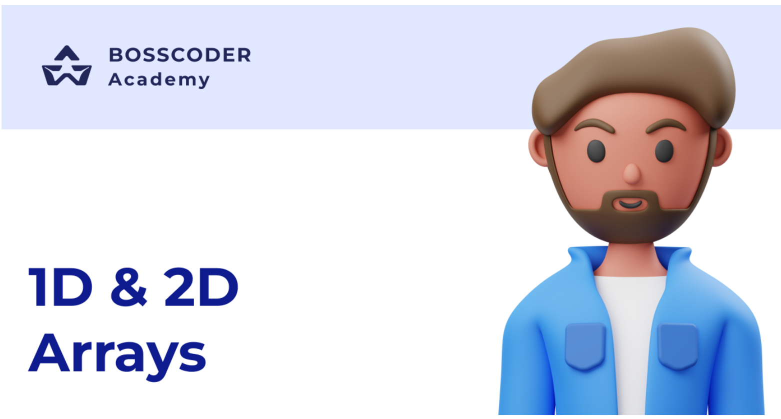
BOSSCODER ACADEMY

UPSKILL WITH US

1D & 2D Arrays

Gyanu Mayank

17th February, 2023



Agenda

1. Problems Based on 1d and 2d Arrays.
2. Various useful techniques such as prefix sum of 1d and 2d Array.

Problems

1. **Sum of all SubMatrices**

   Given an **NxN 2-D** matrix, the task is to find the sum of all the submatrices.

   Example

   Input : N=2, arr[] = {{1, 1}, {1, 1}};

   Output: 16

   Explanation:

   Number of sub-matrices with 1 element = 4

   Number of sub-matrices with 2 elements

= 0

Number of sub-matrices with 4 elements = 1

Since all the entries are 1, the sum becomes sum = 1 * 4 + 2 * 4 + 3 * 0 + 4 * 1 = 16

Brute Force Method

We can solve this question just by traversing the whole matrix N times and generating and calculating their sums. But the problem is this is a very inefficient way to solve this problem. Once, the size of the matrix gets bigger, it will be very time-consuming and start producing TLE. The time complexity of this solution will be **O(n^6)**.

Implementation

```
#include <bits/stdc++.h>
using namespace std;

int main()
{
  int a[10][10];
  int n;
  cin >> n;
  int sum = 0;
  for (int i = 0; i < n; i++)
  {
    for (int j = 0; j < n; j++)
    {
      cin >> a[i][j];
    }
  }
  for (int li = 0; li < n; li++)
```

BOSSCODER ACADEMY UPSKILL WITH US

```
    {
      for (int bi = li; bi < n; bi++)
      {
        for (int bj = lj; bj < n; bj++)
        {

          for (int i = li; i <= bi; i++)
          {
            for (int j = lj; j <= bj; j++)
            {
              sum = sum + a[i][j];
            }
          }
        }
      }
    }
  }
  cout << sum << endl;
}
```

Optimized Approach

The key here is to understand the relation. Let's break it down. We'll first find out, how many times an element of the matrix can occur in submatrices.Let's call this element **arr(x,y)**. Where **x** and **y** is position of element in 0 based indexing.So total occurrence of **arr(x,y)** will be **(x + 1) * (y+ 1) * (n – x) * (n – y)**.Let's call this total number of occurrence be **S(x,y)**.So, total sum of produced by this element will be **arr(x,y) * S(x,y).**Now, we can figure out total sum of submatrices by adding total sum of all the elements of matrix.

Implementation

BOSSCODER ACADEMY UPSKILL WITH US

```
int main()
{

  int a[10][10];
  int n;
  cin >> n;
  int sum = 0;
  for (int i = 0; i < n; i++)
  {
    for (int j = 0; j < n; j++)
    {
      cin >> a[i][j];
    }
  }
  for (int i = 0; i < n; i++)
  {
    for (int j = 0; j < n; j++)
    {

      int top_left = (i + 1) * (j + 1);

      int bottom_right = (n - i) * (n - j);
      sum += (top_left * bottom_right * a[i
    }
  }
  cout << sum << endl;
}
```

Time Complexity: $O(n^2)$
Auxiliary Space: O(1)

2. [Range Sum Queries-1D Array]()

Given an array arr of integers of size n. We and j index values will be executed multiple times.

Example

Input : n=5, arr[] = {1, 2, 3, 4, 5}

i = 1, j = 3

i = 2, j = 4

Output :

9

12

Brute Force Method

We can compute sum for each query independently.

Implementation

```
#include <bits/stdc++.h>
using namespace std;

int main()
{
  int n, q;
  cin >> n >> q;
  int a[n];
  for (int i = 0; i < n; i++)
  {
    cin >> a[i];
  }

  while (q--)
  {
    int x, y;
    cin >> x >> y;
    int sum = 0;
    for (int i = x; i <= y; i++)
    {
```

```
    cout << sum << endl;
  }
}
```

Optimised Approach

An **Efficient Solution** is to precompute prefix sum. Let pre[i] stores sum of elements from arr[0] to arr[i]. To answer a query (i, j), we return pre[j] – pre[i-1].

Implementation

```
#include <bits/stdc++.h>
using namespace std;

int main()
{
  int n, q;
  cin >> n >> q;
  int a[n];
  for (int i = 0; i < n; i++)
  {
    cin >> a[i];
  }
  vector<int> pre(n, 0);
  for (int i = 1; i < n; i++)
  {
    pre[i] = pre[i - 1] + a[i];
  }

  while (q--)
  {
    int x, y;
    cin >> x >> y;
    cout << pre[y] - pre[x - 1] << endl;
```

}

Time Complexity:O(n)
Space Complexity:O(n)

3. **SubMatrix Sum Queries**
Given a matrix of size M x N, there are large number of queries to find submatrix sums. Inputs to queries are left top and right bottom indexes of submatrix whose sum is to find out.
Example
Input: N=4,M=5
mat[N][M] = {{1, 2, 3, 4, 6}, {5, 3, 8, 1, 2}, {4, 6, 7, 5, 5}, {2, 4, 8, 9, 4} };
Query1: tli = 0, tlj = 0, rbi = 1, rbj = 1
Query2: tli = 2, tlj = 2, rbi = 3, rbj = 4
Query3: tli = 1, tlj = 2, rbi = 3, rbj = 3;
Output:
11
38
38

Brute Force Method

For each query, loop over the submatrix and calculate the sum.Each query will be answered in $O(N*M)$. The time complexity of the solution will be $O(Q*N*M)$.With small constraints on N and M, we can answer a handful of queries in a few seconds, but definitely not a million queries.

Implementation

```
#include <bits/stdc++.h>
using namespace std;
```

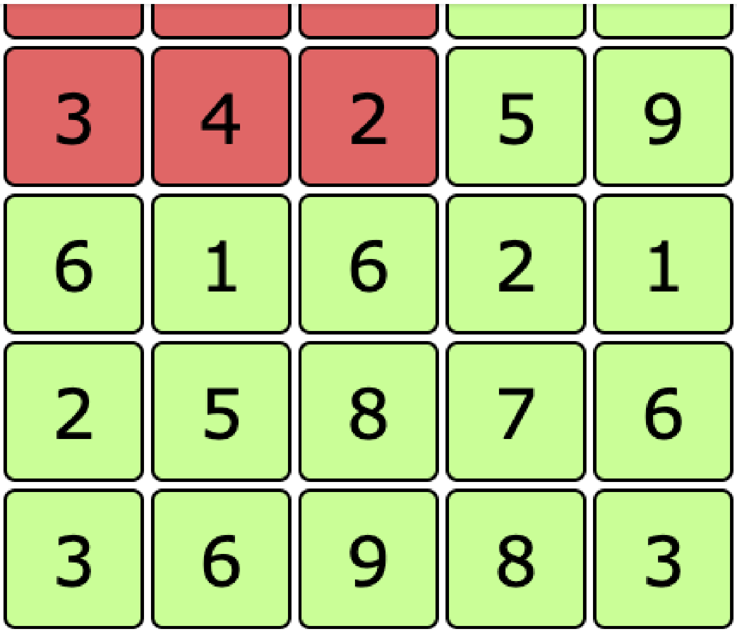
```
  int N, M, Q;
  cin >> N >> M >> Q;
  int A[N][M];
  for (int i = 0; i < N; i++)
    for (int j = 0; j < M; j++)
      cin >> A[i][j];
  while (Q--)
  {
    int x1, y1, x2, y2;
    cin >> x1 >> y1 >> x2 >> y2;

    int sum = 0;
    for (int i = x1; i <= x2; i++)
      for (int j = y1; j <= y2; j++)
        sum += A[i][j];

    cout << sum << "\n";
  }
}
```

Optimised Approach

We can extend the concept of prefix sums in 1-D arrays to 2 dimensions.

Quick recap. The prefix sum in the 1-D array *A* is defined as:pre[i]=pre[i-1]+a[i]

Let's define prefix sum in 2d, pre[i][j] is the sum of element of submatrix with top corner as (1,1) and *bottom right* corner as (*i,j*)



pre[2][3] = A[1][1] + A[1][2] + A[1][3] + A[2][1] + A[2][2] + A[2][3]

sum(x1,x2,y1,y2)=pre[x2][y2]-pre[x2][y1-1]-pre[x1-1][y2]+pre[x1-1][y1-1]

Each query can be answered in $O(1)$ if we have the *pre*[] array.

The *pre*[] array can be computed in $O(N*M)$ in a similar way as above. Here is the formula:

$$pre[i][j]=pre[i][j-1]+pre[i-1][j]-pre[i-1][j-1]+A[i][j]$$

If we iterate the matrix from left to right and top to bottom, at (i,j) we would already have the values at $(i-1,j-1)$,$(i-1,j)$, and$(i,j-1)$. So, calculating the value of $pre[i][j]$ is a constant time operation, i.e., $O(1)$.Pre-computing the entire *pre*[][] matrix takes $O(N*M)$ time

Implementation

```
#include <bits/stdc++.h>
using namespace std;
```

```
{
  int N, M, Q;
  cin >> N >> M >> Q;
  int A[N][M];
  for (int i = 0; i < N; i++)
    for (int j = 0; j < M; j++)
      cin >> A[i][j];
  int pref[N][M];
  for (int i = 0; i < N; i++)
  {
    for (int j = 0; j < M; j++)
    {
      pref[i][j] = A[i][j];
      if (i > 0)
        pref[i][j] += pref[i - 1][j];
      if (j > 0)
        pref[i][j] += pref[i][j - 1];
      if (i > 0 && j > 0)
        pref[i][j] -= pref[i - 1][j - 1];
    }
  }
  while (Q--)
  {
    int x1, y1, x2, y2;
    cin >> x1 >> y1 >> x2 >> y2;

    int sum = pref[x2][y2];
    if (y1 > 0)
      sum -= pref[x2][y1 - 1];
    if (x1 > 0)
      sum -= pref[x1 - 1][y2];
    if (x1 > 0 and y1 > 0)
      sum += pref[x1 - 1][y1 - 1];

    cout << sum << "\n";
```

BOSSCODER ACADEMY

UPSKILL WITH US

Time Complexity:O(N*M)
Space Complexity:O(N*M)

Bosscoder had everything I was looking for

Read

Date: 12th May, 2023

Bosscoder helped me clear my basics of DSA and System Design

Read

Date: 5th May, 2023

View All blogs

Helping ambitious learners upskill themselves & shift to top product

Who are we

About us
Blog
Attend a FREE Event
Privacy Policy
Terms & Condition
Pricing and Refund Policy

Contact Us

Email: ask@bosscoderacademy.com

Follow us on

LinkedIn
Youtube
Instagram
Telegram
Reviews on Quora

BOSSCODER ACADEMY

UPSKILL WITH US

all about
it.