BRACU CP Workshop

Day 2

Shehran Rahman, Shafin Alam

BRAC University, Dhaka

13 April 2025

Prefix Sum

- Let's say we have a one-indexed integer array `arr` of size N and we want to compute the value of

$$\texttt{arr}[a] + \texttt{arr}[a+1] + \cdots + \texttt{arr}[b]$$

for Q different pairs (a, b) satisfying $1 \leq a \leq b \leq N$.

[1] USACO Guide

Prefix Sum

- Let's say we have a one-indexed integer array $\texttt{arr}$ of size N and we want to compute the value of

$$\texttt{arr}[a] + \texttt{arr}[a+1] + \cdots + \texttt{arr}[b]$$

  for Q different pairs (a, b) satisfying $1 \leq a \leq b \leq N$.
- Naively, for every query, we can iterate through all entries from index a to index b to add them up.

[1] USACO Guide

Prefix Sum

- Let's say we have a one-indexed integer array `arr` of size N and we want to compute the value of

$$\texttt{arr}[a] + \texttt{arr}[a+1] + \cdots + \texttt{arr}[b]$$

  for Q different pairs (a, b) satisfying $1 \leq a \leq b \leq N$.
- Naively, for every query, we can iterate through all entries from index a to index b to add them up.
- Since we have Q queries and each query requires a maximum of $\mathcal{O}(N)$ operations to calculate the sum, our total time complexity is $\mathcal{O}(NQ)$. For most problems of this nature, the constraints will be $N, Q \leq 10^5$, so NQ is on the order of 10^{10}. This is not acceptable; it will almost certainly exceed the time limit. [1]

[1] USACO Guide

Prefix Sum

- We designate a prefix sum array `prefix`. First, because we're 1-indexing the array, set $\texttt{prefix}[0] = 0$, then for indices k such that $1 \leq k \leq N$, define the prefix sum array as follows:

$$\texttt{prefix}[k] = \sum_{i=1}^{k} \texttt{arr}[i]$$

[2] USACO Guide

Prefix Sum

- We designate a prefix sum array `prefix`. First, because we're 1-indexing the array, set $\texttt{prefix}[0] = 0$, then for indices k such that $1 \leq k \leq N$, define the prefix sum array as follows:

$$\texttt{prefix}[k] = \sum_{i=1}^{k} \texttt{arr}[i]$$

- Now, when we want to query for the sum of the elements of `arr` between (1-indexed) indices a and b inclusive, we can use the following formula:

$$\sum_{i=L}^{R} \texttt{arr}[i] = \sum_{i=1}^{R} \texttt{arr}[i] - \sum_{i=1}^{L-1} \texttt{arr}[i]$$

[2]

[2] USACO Guide

Static Range Sum Query

- Using our definition of the elements in the prefix sum array, we have

$$\sum_{i=L}^{R} \texttt{arr}[i] = \texttt{prefix}[R] - \texttt{prefix}[L-1]$$

[3]

[3] USACO Guide

Problem 1

Ilya and Queries

- **Given**:
  - String s of length n $(2 \leq n \leq 10^5)$
  - Characters are either '.' or '#'
  - m queries $(1 \leq m \leq 10^5)$
- **Query**:
  - Two indices l_i, r_i $(1 \leq l_i < r_i \leq n)$
  - Count indices i where $s[i] = s[i+1]$ in $[l_i, r_i)$
- **Output**:
  - Answer all queries in order

Difference Array Technique

Efficient Range Updates

- **Purpose**:
  - Range Update: add x to all values in range $[l, r]$
  - Perform range updates in constant time
  - Convert to final array in linear time

Difference Array Technique

Efficient Range Updates

- **Purpose**:
  - Range Update: add x to all values in range $[l, r]$
  - Perform range updates in constant time
  - Convert to final array in linear time
- **Steps**:
  - For update $[l, r] + val$:
    - $D[l] += val$
    - $D[r+1] -= val$
  - Compute final array using prefix sum technique

Problem 2

Little Girl and Maximum Sum

- **Given**:
  - Array A of n elements $(1 \leq n \leq 2 \times 10^5)$
  - q queries $(1 \leq q \leq 2 \times 10^5)$
- **Query**:
  - Range $[l_i, r_i]$
  - Returns sum $A[l_i] + A[l_i + 1] + \cdots + A[r_i]$
- **Challenge**:
  - Reorder array elements to maximize total sum of all query answers

Problem 3

Greg and Array

- **Given**:
  - Initial array A of size n $(1 \leq n \leq 10^5)$
  - m range operations $(1 \leq m \leq 10^5)$
  - k queries $(1 \leq k \leq 10^5)$
- **Operations**:
  - Operation i: Add d_i to $A[l_i..r_i]$
- **Queries**:
  - Query j: Apply operations x_j to y_j
- **Output**:
  - Final array after all queries

2D Prefix Sum

- In two dimensional prefix sum array

$$\texttt{prefix}[a][b] = \sum_{i=1}^{a} \sum_{j=1}^{b} \texttt{arr}[i][j].$$

2D Prefix Sum

- In two dimensional prefix sum array

$$\texttt{prefix}[a][b] = \sum_{i=1}^{a} \sum_{j=1}^{b} \texttt{arr}[i][j].$$

- This can be calculated as follows for row index $1 \le i \le n$ and column index $1 \le j \le m$:

$$\begin{aligned}\texttt{prefix}[i][j] = & \texttt{prefix}[i-1][j] + \texttt{prefix}[i][j-1] \\ & - \texttt{prefix}[i-1][j-1] + \texttt{arr}[i][j]\end{aligned}$$

2D Prefix Sum

- The submatrix sum between rows a and A and columns b and B, can thus be expressed as follows:

$$\sum_{i=a}^{A}\sum_{j=b}^{B}\texttt{arr}[i][j] = \texttt{prefix}[A][B] - \texttt{prefix}[a-1][B] \\ - \texttt{prefix}[A][b-1] + \texttt{prefix}[a-1][b-1]$$

Problem 4

Convolution [Problem H]

- **Given**:
  - Input matrix I of size $n \times m$ $(1 \leq n, m \leq 10^3)$
  - Kernel size $k \times l$ $(1 \leq k \leq n,\ 1 \leq l \leq m)$
  - Kernel elements restricted to $-1, 0, 1$
- **Operation**:
  - 2D convolution:
    $O(p, q) = \sum_{x=1}^{k} \sum_{y=1}^{l} K(x, y) \times I(p + x - 1, q + y - 1)$
  - Output matrix size: $(n - k + 1) \times (m - l + 1)$
- **Challenge**:
  - Find the maximum sum of all elements in the output matrix O among all possible matrix K

Problem 5

Running Miles

- **Scenario**:
  - Street with n sights at positions $1, 2, \ldots, n$ miles
  - Sight at position i has beauty b_i
- **Task**:
  - Choose jog interval $[l, r]$ with at least 3 sights
  - Maximize: sum(3 max beauties) $-$ $(r - l)$
- **Constraints**:
  - $t \leq 10^5$ test cases
  - $n \leq 10^5$ per test case
  - $\sum n \leq 10^5$
  - $1 \leq b_i \leq 10^8$

Stack

Basic Operations

- `push(x)` : Push an item on top of the stack.
  `pop()` : Pop the top-most element from stack.
  `top()` : Returns the top-most value from stack.
  `empty()` : Returns true if stack is empty.

Queue

Basic Operations

- `push(x)` : Push an item at the end of the queue.
  `pop()` : Pop the first entry from the queue.
  `front()` : Returns the front-most element of the queue.
  `empty()` : Returns true if queue is empty.

Queue

Basic Operations

- `push(x)` : Push an item at the end of the queue.
  `pop()` : Pop the first entry from the queue.
  `front()` : Returns the front-most element of the queue.
  `empty()` : Returns true if queue is empty.
- There is also a DS known as deque (Double Ended Queue). Simply allows entry/removal from both side of queue.
  `push_back(x)`, `push_front(x)`, `pop_back(x)`, `pop_front(x)`, `front()`, `back()`, `empty()` - all behave as you would expected.

Monotone stack

- Given an array, a, of N $(1 \leq N \leq 10^5)$ integers, for every index i, find the nearest index j to the left such that $j < i$ and $a_j < a_i$.

Monotone stack

- What is the time complexity of our solution?

Sliding Window

- A sliding window is a constant-size subarray that moves from left to right through the array.
- At each window position, we want to calculate some information about the elements inside the window.

Sliding Window Minimum

- Given an array A of size n. For each k-size Window, find the smallest element inside the window.

Sliding Window Minimum

Step 0:	2	1	4	5	3	4	1	2
Step 1:	2	1	4	5	3	4	1	2
Step 2:	2	1	4	5	3	4	1	2
Step 3:	2	1	4	5	3	4	1	2
Step 4:	2	1	4	5	3	4	1	2
Step 5:	2	1	4	5	3	4	1	2

Sliding Window Minimum

- To find the sliding window minimum, we use a queue that always keeps the smallest element at the front. All elements in the queue are in non-decreasing order, so the front is always the minimum in the current window.

Sliding Window Minimum

- To find the sliding window minimum, we use a queue that always keeps the smallest element at the front. All elements in the queue are in non-decreasing order, so the front is always the minimum in the current window.

[4] Antti Laaksonen, *Competitive Programmer's Handbook*, Chapter 8.3, 2018.

Sliding Window Minimum

- To find the sliding window minimum, we use a queue that always keeps the smallest element at the front. All elements in the queue are in non-decreasing order, so the front is always the minimum in the current window.
- After each window move, we remove elements from the end of the queue until the last queue element is smaller than the new window element, or the queue becomes empty. We also remove the first queue element if it is not inside the window anymore. Finally, we add the new window element to the end of the queue. [4]

[4] Antti Laaksonen, *Competitive Programmer's Handbook*, Chapter 8.3, 2018.

Time Complexity

- What is the time complexity of our solution

Time Complexity

- What is the time complexity of our solution
- The time complexity of our solution is $\mathcal{O}(n)$,

Problem 6

Maximum Subarray Sum

- Given an array of n integers.
- Find the maximum sum of any contiguous, nonempty subarray.

Problem 7

Maximum Subarray Sum II

- You are given an array of n integers, and two integers a and b $(1 \leq a \leq b \leq n)$.
- Find the maximum sum of values in any contiguous subarray whose length is between a and b (inclusive).

Set Data Structure

- A balanced BST that offers some quick results as follows. We will discuss it more in depth in the STL class.

Set Data Structure

- A balanced BST that offers some quick results as follows. We will discuss it more in depth in the STL class.
- `insert(x)` : Insert a value in x in $\mathcal{O}(\lg n)$.
  `erase(x)`: Erase a value x (if exists) in $\mathcal{O}(\lg n)$.
  `find(x)`: Returns the location/iterator to x if it exists in the ds in $\mathcal{O}(\lg n)$.

Set Data Structure

- A balanced BST that offers some quick results as follows. We will discuss it more in depth in the STL class.
- `insert(x)` : Insert a value in x in $\mathcal{O}(\lg n)$.
  `erase(x)`: Erase a value x (if exists) in $\mathcal{O}(\lg n)$.
  `find(x)`: Returns the location/iterator to x if it exists in the ds in $\mathcal{O}(\lg n)$.
- A key property of set is it always keeps everything sorted inside (which is how it achieves the faster time complexity of above operations).

Set Data Structure

- A balanced BST that offers some quick results as follows. We will discuss it more in depth in the STL class.
- `insert(x)` : Insert a value in x in $\mathcal{O}(\lg n)$.
  `erase(x)`: Erase a value x (if exists) in $\mathcal{O}(\lg n)$.
  `find(x)`: Returns the location/iterator to x if it exists in the ds in $\mathcal{O}(\lg n)$.
- A key property of set is it always keeps everything sorted inside (which is how it achieves the faster time complexity of above operations).
- The standard set keeps only distinct entries. There is another variant known as `multiset` which allows duplicate values to exists.

Problem 8

Sliding Window Median

- You are given an array, a, of n ($1 \leq n \leq 10^5$) integers, and an integer k ($1 \leq k \leq n$).
- Your task is to calculate the median of each contiguous subarray (window) of size k, from left to right.

Problem 9

Merging Arrays

- You are given two arrays, sorted in non-decreasing order. Merge them into one sorted array.

Problem 10

Books

- Valera has n books and t free minutes.
- The i-th book takes a_i minutes to read.
- He can choose a starting index i and read books $i, i+1, \ldots$ in order, stopping if he doesn't have enough time to fully read the next book.
- Find the maximum number of books he can read.

Problem 11

Segments with Small Set

- You are given an array a of n integers and an integer k.
- A segment $a[l..r]$ $(1 \leq l \leq r \leq n)$ is *good* if it contains at most k distinct elements.
- Count the number of good segments.

Problem 12

Subarray Sums I

- You are given an array of n positive integers, and an integer x.
- Count the number of subarrays whose sum is equal to x.

Problem 13

Sum of Two Values

- You are given an array, a, of n $(1 \leq n \leq 10^5)$ integers, and an integer x. Find two indices i and j such that $i \neq j$ and $a_i + a_j = x$.

Problem 14

Count all triplets with given sum in sorted array

- Given a sorted array, a, of n $(1 \leq n \leq 10^5)$ integers, and a target value x.
- Count the number of triplets (i, j, k) such that $i < j < k$ and $a_i + a_j + a_k = x$.

Problem 15

Segments with Small Spread

- Given an array of n integers a_i, and an integer k.
- A segment $a[l..r]$ $(1 \leq l \leq r \leq n)$ is *good* if the difference between the maximum and minimum elements in the segment is at most k.
- Your task is to count the number of different good segments.