Optimization

Christy Cheng {christycty}

2021-02-27

Motivation

C++11	Time Limit Exceeded
C++11	Time Limit Exceeded
C++11	Time Limit Exceeded
C++11	Time Limit Exceeded
C++11	Time Limit Exceeded
C++11	Time Limit Exceeded
C++11	Time Limit Exceeded
C++11	Time Limit Exceeded
C++11	Time Limit Exceeded

- We need to reduce time complexity by **avoiding repeated computation**

$O(N!)$ → $N \leq 10$

$O(2^N)$ → $N \leq 20$

$O(N^2)$ → $N \leq 5000$

$O(N \lg N)$ → $N \leq 10^6$

$O(N)$ → $N \leq 10^7$

Outline

- ❏ Partial Sum
- ❏ Difference Array
- ❏ Precomputation
- ❏ Sliding Window (Two Pointers)
- ❏ Batching Step
- ❏ Discretization

1D Partial Sum - Problem

Given an array of integers and some queries.

For each query, find the sum of a contiguous section of the array.

i	1	2	3	4	5	6	7	8	9
a[i]	3	-2	6	1	0	5	-5	2	4

Example: sum of 3rd to 6th element = a[3] + a[4] + a[5] + a[6]

= 6 + 1 + 0 + 5

= 12

1D Partial Sum - Naive Solution

```
for i ← 1 to N do
    input a[i]

for i ← 1 to Q do
    input L and R
    sum = 0
    for j ← L to R do
        sum += a[j]
    output sum
```

Notations:

N = size of array

Q = number of queries

L = start of contiguous section

R = end of contiguous section

Time Complexity?

O(NQ)

(Imagine all queries ask for a[1] to a[N])

1D Partial Sum - Naive Solution

i	1	2	3	4	5	6	7	8	9
a[i]	3	-2	6	1	0	5	-5	2	4

We can see that the sum of a[4] to a[6] is calculated many times...

Can we perform the calculation once only?

1D Partial Sum - Idea

`c[i]` = sum of the first i elements (a[1]+a[2]+...+a[i])

i	1	2	3	4	5	6	7	8	9
a[i]	3	-2	6	1	0	5	-5	2	4
c[i]	3	1	7	8	8	13	8	10	14

To find the sum from a[L] to a[R], calculate c[R] - c[**L-1**]

e.g. sum of a[3] to a[5] = c[5] - c[2] = 7

c[5] = a[1] + ... + a[5], c[2] = a[1] + a[2] → c[5] - c[2] = a[3] + a[4] + a[5]

1D Partial Sum - Implementation

```
for i ← 1 to N do
    c[i] = c[i-1] + a[i]

for i ← 1 to Q do
    input L and R
    output c[R]-c[L-1]
```

Time Complexity

O(N) partial sum precomputation

O(1) per query → total O(Q)

= O(N + Q)

Reminder:

- Use long long if range of a[i] is large
- Initialize c[0] first
- Be aware of 0/1-based issue

2D Partial Sum - Problem

Now, let's extend the question to 2-dimension

-3	1	2	0	2
2	0	4	-3	1
-1	3	0	1	4
5	-2	1	4	2

Given an array a with N rows and M columns and some queries.

In each query, you need to find the sum of elements from (Sx, Sy) to (Ex, Ey).

(x, y) = element at x^{th} row and y^{th} column

e.g. circled area is from (1, 2) to (2, 4) → ans = 4

2D Partial Sum - Naive Solution

```
for i ← 1 to Q do
    input Sx, Sy, Ex, Ey
    sum = 0
    for j ← Sx to Ex do
        for k ← Sy to Ey
    do
            sum += a[j][k]
    output sum
```

Loop over the required area in each query and sum up the numbers

Time Complexity?

O(NMQ)

Imagine all queries ask for (1, 1) to (N, M)

2D Partial Sum - 1D Optimized Solution

Compute a partial sum array for each row

-3	1	2	0	2
2	0	4	-3	1
-1	3	0	1	4
5	-2	1	4	2

-3	-2	0	0	2
2	2	6	3	4
-1	2	2	3	7
5	3	4	8	10

2D Partial Sum - 1D Optimized Solution

```
for i ← 1 to N do
    for ← 1 to M do
        c[i][j] = c[i][j-1] + a[i][j]

for i ← 1 to Q do
    input Sx, Sy, Ex, Ey
    sum = 0
    for j ← Sx to Ex do
        sum += c[j][Ey] - c[j][Sy-1]
    output sum
```

Time Complexity

O(NM) partial sum precomputation

O(min(N, M)) per query

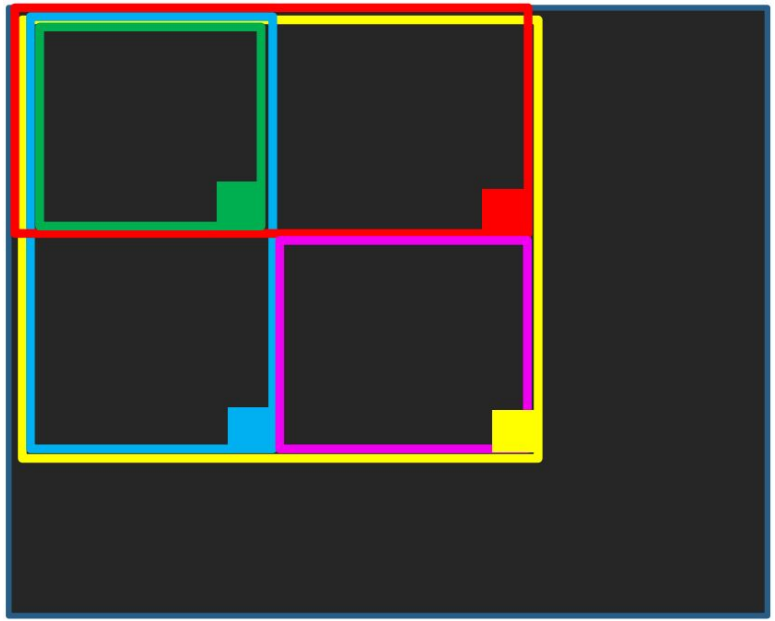
= O(NM + min(N,M) * Q)

Not bad, but can it be better?

(esp when Q is very large)

2D Partial Sum - Idea

Magenta = Yellow – Red – Blue + Green



1	2
3	4

1 + 2 + 3 + 4

- 1 - 2

- 1 - 3

+ 1

= 4

2D Partial Sum - Idea

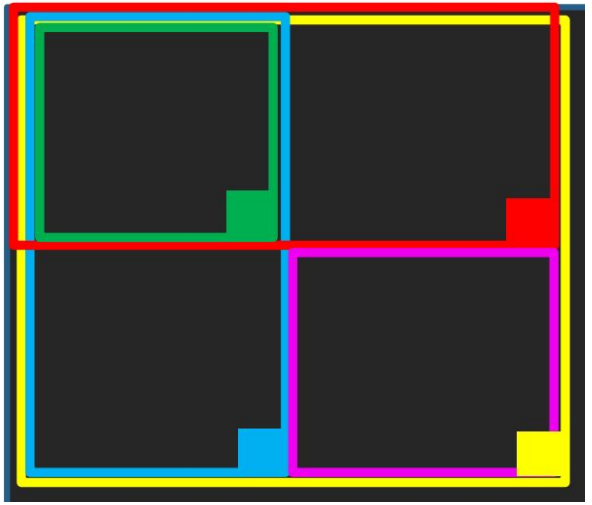
How to compute area of each region?

from last slide, we have:

Magenta = Yellow – Red – Blue + Green

after rearranging some terms, we have:

Yellow = Magenta + Red + Blue – Green

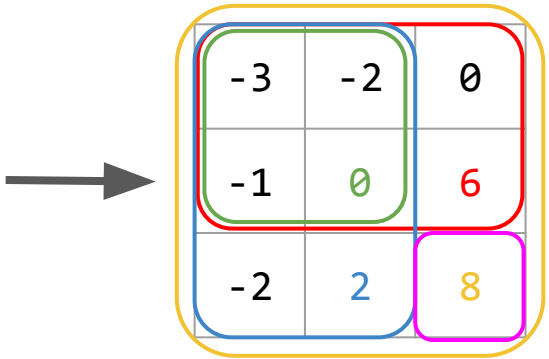


2D Partial Sum - Precomputation

Let `c[i][j]` = sum of elements from `a[1][1]` to `a[i][j]`

```
for i ← 1 to N do
for j ← 1 to M do
    c[i][j] = a[i][j] + c[i-1][j] + c[i][j-1] - c[i-1][j-1]
```

-3	1	2
2	0	4
-1	3	0



2D Partial Sum - Query

```
for i ← 1 to Q do
    input Sx, Sy, Ex, Ey
    sum = c[Ex][Ey] - c[Sx-1][Ey] - c[Ex][Sy-1] + c[Sx-1][Sy-1]
    output sum
```

Time Complexity

O(NM) precomputation and O(1) per query

= O(NM + Q)

2D Partial Sum - Example

Query (2, 3) to (5, 4)

c[5][4] - c[1][4] - c[5][2] + c[1][2]

= 11 - 0 - 8 + (-2)

= 1

-3	1	2	0	2
2	0	4	-3	1
-1	3	0	1	4
4	2	-2	1	3

-3	-2	0	0	2
-1	0	6	3	6
-2	2	8	6	13
2	8	12	11	21

2D Partial Sum - Example

Query (2, 2) to (3, 5)

c[4][5] - c[1][5] - c[3][1] + c[1][1]

= 13 - 2 - (-2) + (-3)

= 10

-3	1	2	0	2
2	0	4	-3	1
-1	3	0	1	4
4	2	-2	1	3

-3	-2	0	0	2
-1	0	6	3	6
-2	2	8	6	13
2	8	12	11	21

1D Difference Array - Problem

Given some actions. In each action, add v_i to a contiguous section of the array. Find the final value of the array.

Add 2 to `a[2..4]`

i	1	2	3	4	5	6
a[i]	0	2	2	2	0	0

Add 3 to `a[3..6]`

i	1	2	3	4	5	6
a[i]	0	2	5	5	3	3

1D Difference Array - Naive Solution

```
for i ← 1 to P do
    input L, R, V
    for j ← L to R do
        a[j] += V

for i ← 1 to N do
    output a[i]
```

Notations:

N = size of array

P = number of actions

L = start of contiguous section

R = end of contiguous section

Time Complexity?

O(PN)

(Imagine all actions act on a[1] to a[N])

1D Difference Array - Idea

Assume the actions are:

1. add **2** for a[2] to a[6]
2. add **3** for a[4] to a[8]
3. add **4** for a[7] to a[7]

Final Result:

i	1	2	3	4	5	6	7	8	9
a[i]	0	2	2	5	5	5	7	3	0

difference **+2** 0 **+3** 0 0 +2 **-4** **-3**

(+2 = **+4** **-2**)

1D Difference Array - Idea

Instead of updating the whole contiguous section, we can just mark the value at start and end points using array `d`.

array `a` can be computed based on d after all actions are done.

i	1	2	3	4	5	6	7	8	9
d[i]		+2		+3			-2 +4	-4	-3
a[i]	0	2	2	5	5	5	7	3	0

- add **2** for a[2] to a[6]
- add **3** for a[4] to a[8]
- add **4** for a[7] to a[7]

- Note that we are marking `d[R+1]` instead of `d[R]`

1D Difference Array - Implementation

```
for i ← 1 to P do
    input L, R, V
    d[L] += V
    d[R+1] -= V

for i ← 1 to N do
    a[i] = a[i-1] + d[i]
    output a[i]
```

Time Complexity
= O(N+P)

2D Difference Array - Problem

Now, let's extend the question to 2-dimension

0	2	2	2	0
0	2	2	2	0
0	0	0	0	0
0	0	0	0	0

Given an array a with N rows and M columns and some actions.

In each action, you need to add v_i all to elements from (Sx, Sy) to (Ex, Ey).

example:

add 2 to all elements from (1, 2) to (2, 4)

2D Difference Array - Idea

Define an array d such that

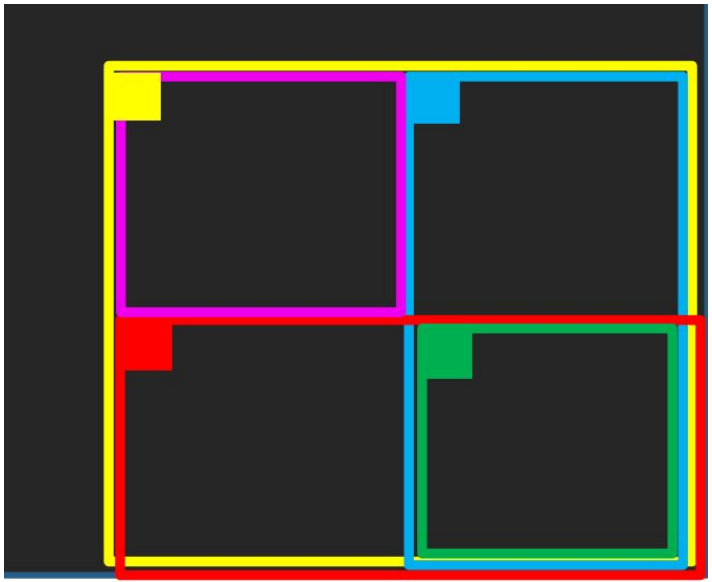
```
d[i][j] = a[i][j] - a[i][j-1] - a[i-1][j] + a[i-1][j-1]
```

0	2	2	4
2	5	2	3
2	6	1	5
3	4	0	1

0	2	0	2
2	1	-3	-1
0	1	-2	3
1	-3	1	-3

2D Difference Array - Idea

This picture again... but this time rotated



Imagine the small filled squares as `d[i][j]`, and the frame is the area affected by `d[i][j]`.

To update [Sx...Ex, Sy...Ey], we first add v to **d[Sx][Sy]**.

Then, we block the increase outside the area by subtracting v from **d[Sx][Ey+1]** and **d[Ex+1][Sy]**.

Since v is subtracted twice from [Ex+1...N, Ey+1...M], we add v to **d[Ex+1][Ey+1]**.

2D Difference Array - Idea

Now, let's see how we can compute array `a` from `d`.

In the previous slides, we defined:

```
d[i][j] = a[i][j] - a[i][j-1] - a[i-1][j] + a[i-1][j-1]
```

By rearranging the equation, we get:

```
a[i][j] = d[i][j] + a[i][j-1] + a[i-1][j] - a[i-1][j-1]
```

2D Difference Array - Action

```
for i ← 1 to P do
    Input Sx, Sy, Ex, Ey
    d[Sx][Sy] += v
    d[Sx][Ey+1] -= v
    d[Ex+1][Sy] -= v
    d[Ex+1][Ey+1] += v
```

0	0	0	0
5	5	5	0
5	5	5	0
0	0	0	0

0	0	0	0
+5	0	0	-5
0	0	0	0
-5	0	0	+5

2D Difference Array - Compute Result

```
for i ← 1 to N do
    for j ← 1 to M do
        a[i][j] = d[i][j] + a[i][j-1] + a[i-1][j] - a[i-1][j-1]
        output a[i][j]
```

Time Complexity = O(NM + P)

Precomputation - Problem

Given a string that consists of ‘A’, 'B' and some queries.

For each query qi, output the closest ‘B’ whose index <= qi.

Example:

string = “AABAAABBA”

query 3 → output 3

query 6 → output 3

query 9 → output 8

Precomputation - Naive solution

```
for i ← 1 to Q do
    input qi
    for j ← qi downto 1 do
        if (s[j] = 'B')
            output j
            break
```

Notations:

s = given string

N = length of string

Time complexity = O(QN)

- Remember to handle the case where 'B' is not found

Precomputation - Solution

Build an array `a` where `a[i]` means the last “B” which index <= `i`

For example, string = “AABAAABBA”:

i	1	2	3	4	5	6	7	8	9
a[i]	-1	-1	3	3	3	3	7	8	8

- `-1` is used to indicate that there is no ‘B’ with index <= `i`.

Precomputation - Solution

```
a[0] = -1
for i ← 1 to N do
    if (s[i] = ‘B’)
        a[i] = i
    else
        a[i] = a[i-1]

for i ← 1 to Q do
    input qi
    output a[qi]
```

Total time complexity = O(N + Q)

Precomputation

Some useful arrays to be precomputed

- ❏ Prefix / suffix sum (partial sum)
- ❏ Prefix / suffix max / min / special element
- ❏ Prefix / suffix XOR sum
- ❏ Prefix / suffix index of last special element
- ❏ Prefix / suffix count (e.g. number of odd numbers / number of “*”s)

Practice

20108 Maximum Sum

I0111 Mobile Phones

J051 Pattern Matching

J134 Lucky Rainbow

M1513 Advanced Optimization Coding

S113 Gravity Game

S164 Alice's Meal

T062 Tappy World

Example - 20108 Maximum Sum

```
for i ← 1 to N do //Sx
for j ← 1 to N do //Sy
for k ← i to N do    //Ex
for l ← j to N do //Ey
    sum = 0
    for x ← i to k do
    for y ← j to l do
        sum += a[x][y]
    ans = max(sum, ans)
```

Fixing the upper left and bottom right corners of the rectangle

Instead, apply 2D partial sum
→ from O(N^2) to O(1)

Example - T062 Tappy World

- similar to our 1D difference array example!

```
for i ← 1 to Q do
    input D, X, Y
    d[X] += D
    d[Y+1] -= D

for i ← 1 to N do
    a[i] = a[i-1] + d[i]
```

Sliding Window - Problem

Given two sorted array `a` and `b`,

find the number of pair (i, j) such that `a[i] + b[j] = C`

i	1	2	3	4	5	6	7	8	9
a[i]	1	5	8	10	12	14	15	18	25
b[i]	3	6	6	7	13	14	15	18	19

If C = 20, ans = 4 [(1, 9), (2, 7), (6, 2), (6, 3)]

Sliding Window - Naive Solution

```
for i ← 1 to N do
    for j ← 1 to N do
        if (a[i] + b[j] = C)
            ans += 1
output ans
```

Exhaust all pairs of (i, j) and count how many of them satisfy `a[i] + b[j] = C`

Time Complexity = $O(N^2)$

Hint: the arrays are **sorted**

Sliding Window - Binary Search Solution

For each element in a,
binary search the number of `b[j]` such that `b[j] = C - a[i]`

Two binary search (one for lower bound, one for upper bound) are needed if the elements in `b` are not distinct.

Time complexity = O(N log N)

Sliding Window - Observation

1	5	8	10	12	14	15	18	25
3	6	6	7	13	14	15	18	19

1	5	8	10	12	14	15	18	25
3	6	6	7	13	14	15	18	19

1	5	8	10	12	14	15	18	25
3	6	6	7	13	14	15	18	19

1	5	8	10	12	14	15	18	25
3	6	6	7	13	14	15	18	19

i increases
→ a[i] increases
→ C - a[i] decreases
→ b[j] decreases
→ j decreases

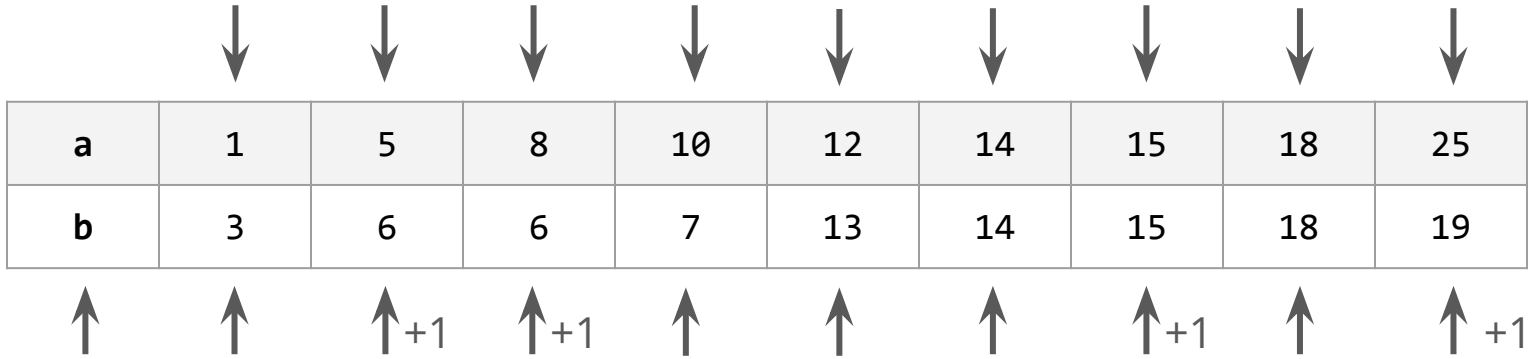
Sliding Window - Implementation

```
j = N
for i ← 1 to N do
    while (j > 0) and (a[i] + b[j] > C)
        j -= 1
    while (j > 0) and (a[i] + b[j] = C)
        ans += 1
        j -= 1
output ans
```

Time Complexity = O(N)

Sliding Window - Demonstration

Find number of pairs (i, j) such that a[i] + b[j] = C. Assume C = 20.




```
for i ← 1 to N do
    while (j > 0) and (a[i] + b[j] > C) j -= 1
    while (j > 0) and (a[i] + b[j] = C) ans++, j -= 1
```

Sliding Window - M0652 Museum

Given N, M and an array of integers a.

Find the shortest consecutive segment that contains all numbers 1..M.

Example:

N = 12, M = 5

2	5	3	1	3	2	4	1	1	5	4	3

Sliding Window - M0652 Museum

We can set the start and end of the segment as our two pointers.

```
j = 1
for i ← 1 to N do
    while (a[i..j] does not contain all m integers) do
        j += 1
    ans = min(ans, j - i + 1)
```

- ➢ Counting array can be used to maintain if current segment contains all m integers.
- ➢ Remember to handle cases when `j >= N`.

- Note that `j` would not decrease when `i` increases.

Sliding Window - M0652 Museum

2	5	3	1	3	2	4	1	1	5	4	3

2	5	3	1	3	2	4	1	1	5	4	3

2	5	3	1	3	2	4	1	1	5	4	3

2	5	3	1	3	2	4	1	1	5	4	3

2	5	3	1	3	2	4	1	1	5	4	3

2	5	3	1	3	2	4	1	1	5	4	3

Sliding Window - Application

- Input is two array or one array that references itself

- the arrays are sorted, or
- the things we want to find have monotonicity (e.g. sum / count of sth)

Batching Step

- In some simulation problems, some continuous steps can be performed at once.

Find the date of now + N days

➢ Instead of iterating over days, iterate over months
➢ Deduct Mi days from N

J184 Mysterious Area

➢ Instead of moving grid by grid horizontally, directly move to the nearest pillar taller than or equal to current height

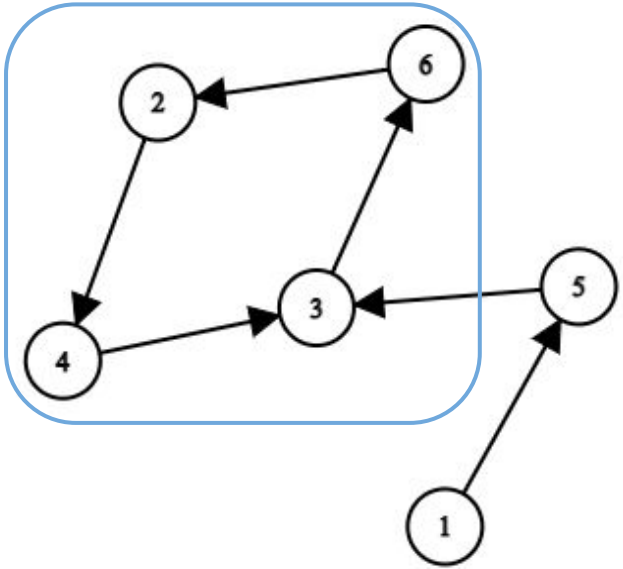
Batching Step - Find Cycles

In some simulation problems, we may encounter cycles that are traversed many times, causing TLE.

Walk X ($1 \le X \le 10^{18}$) steps starting from node 1, what is the final position?

If we simulate each step (O(X)),

we will get TLE :<



Batching Step - Find Cycles

Instead, we can skip the steps that traverse the whole cycle.

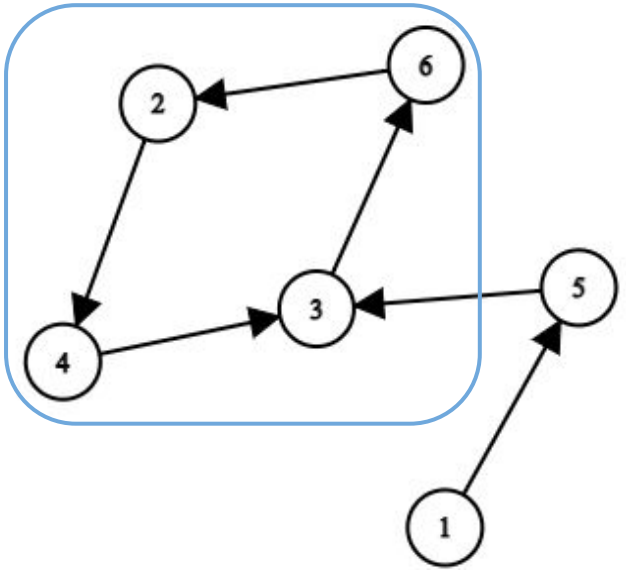
(The results won't be affected by the number of cycles traversed)

The simulation will then become:

2 steps (1 → 5 → 3)

\+ (X - 2) % 4 steps

The time complexity is now O(N)



Discretization - Example

Count the number of occurrence of some numbers in array `a` (`a[i]` <= 10^9)

whereas size of `a` <= `100000`

The most straightforward solution would be using a counting array.

but... we cannot create a counting array of [1..10^9] :/

Discretization - Idea

- Discretization (離散法) is a technique that converts values (not necessarily integers) into integers, while maintaining their relative order
- Example: 7654321, 123456, 934602, 123456789 → 3, 1, 2, 4 (or 2, 0, 1, 3)
- Put the values into an array, sort the array 123456, 934602, 7654321, 123456789
- For each value in the original array, find its rank using binary search *(or you can use map for this step, more in C++ STL / data structure)*
- Note: it's better for the same value to be converted into the same number *(by making the sorted array unique)*

Discretization - Solution

Count the number of occurrence of some numbers in array `a` (`a[i]` <= 10^9)

We can perform discretization and map the numbers to their rank.

A counting array can be built with size = number of **unique** numbers.

During counting/query, we can use the value's rank to access the counting array to modify/retrieve it.

More Practice

J144 Fair Santa Claus

M0652 Museum

M1613 Fair Patrol

S072 Partners

S152 Apple Garden

S211 Skyscraperhenge

T042 Game of Life IV

Special Thanks

past optimization slides: cylau, jeremy624

checking and modification: yaufung

practice tasks: dbschi

Q&A *(if any)*