J243 - Neat Corridor

Kelvin Chow {Lrt1088}

2024-02-17

Background

Problem idea by kctung

Preparation by trashcan, gabrielliu2001, christycty, happychau

Problem Restatement

Given A[N], a sequence of 0 and 1 with length N and a constant K, apply minimal amount of the following operations:

- For any position 1 <= i <= N - K + 1, flip all the value in range A[i] to A[i + K - 1] from 0 to 1 and vice versa.

So that all the 0 in the resulting sequence goes before 1, or report if there is no solution.

Example

Neat sequences: `0001`, `0111`, `0000`, `1111`

Not a neat sequence: `0101`, `1000`

Sample 1:

```
5 3        i=1       i=3                              length = 2
10100 -> 01000 -> 01111 (neat)

5 3        i=1       i=2       i=3                    length = 3
10100 -> 01000 -> 00110 -> 00001 (neat)
```

Statistic

0 points	30	+ 0 +	5	+ 1 =	31
8 points	2	+ 1 +	0	+ 0 =	3
14 points	1	+ 0 +	0	+ 0 =	0
17.5 points	5	+ 4 +	1	+ 0 =	9
20 points	0	+ 0 +	1	+ 0 =	1
21.5 points	6	+ 6 +	1	+ 1 =	14
23.5 points	1	+ 1 +	1	+ 0 =	3
27.5 points	0	+ 2 +	0	+ 0 =	2
30 points	0	+ 0 +	1	+ 0 =	1
35 points	1	+ 2 +	3	+ 2 =	8
39 points	0	+ 2 +	3	+ 0 =	5
40 points	0	+ 0 +	1	+ 0 =	1
50 points	0	+ 0 +	1	+ 0 =	1
52.5 points	0	+ 0 +	0	+ 1 =	1
54 points	0	+ 0 +	0	+ 2 =	2
62 points	0	+ 0 +	1	+ 0 =	1
67.5 points	0	+ 0 +	0	+ 1 =	1

Attempts	Max	Mean	Std Dev
65	67.500	24.092	15.621

Subtasks						
8: 38 4: 18	12: 13 6: 40	15: 9 7.5: 38	10: 2 5: 17	17: 1 8.5: 16	8: 0 4: 14	30: 0 15: 4

Subtasks

For all subtasks, you score 50% if your program can:

- detect invalid cases
- give a valid solution to valid cases with length not exceeding 10^6.

SUBTASKS

For all cases:
$1 \le N \le 100000$
$1 \le K \le N$

	Points	Constraints
1	8	$N = 3, K = 1$
2	12	$N \le 200, K = 1$
3	15	$K = 1$
4	10	$N \le 200$
5	17	$N \le 5000$
6	8	$K = 2$
7	30	No additional constraints

Before we start

If you haven't notice, the operation is xor 1 through a consecutive sequence.

Thus we have the following properties. Let a, b, c be operations performed on different position:

1. They are commutative. This means, $a \wedge b = b \wedge a$.
2. They are associative. This means, $(a \wedge b) \wedge c = a \wedge (b \wedge c)$.
3. They are self-inverse. This means, any operation performed twice at the same position is equal to do nothing. $a \wedge a = 0$.

Therefore the focus of this problem is selecting minimal needed positions to perform operations on.

Subtask 1-3 Partial Scores

Subtask 1-3 (35%): (some modifiers), K = 1

Since a resultant sequence with all 0s is a valid sequence, you only need to output the positions that are 1s in the sequence.

Score: 35% * 50% = 17.5%

Time Complexity: O(N)

Subtask 1

Subtask 1 (8%): N = 3, K = 1

- Sanity check. There are only $2^3 = 8$ possible cases.
- Store all the answers and output correspondingly.

000: 0
001: 0
010: 1 (2 or 3)
011: 0
100: 1 (1)
101: 1 (1 or 2)
110: 1 (3)
111: 0

Score: 8

Time Complexity: O(1)

Subtask 2

Subtask 2 (12%): N <= 200, K = 1

- There are only N + 1 possible resulting sequences.
  - 00...00, 00...01, 00...11, ..., 01...11, 11...11
- Enumerate through each of them and determine the minimum operations needed.
  - Count the positions that differ from the desired sequence.

Score: 12 (Cumulative: 20)

Time Complexity: O(N^2)

Subtask 3

Subtask 3 (15%): K = 1

- Recall the approach from subtask 2:
  - Count the positions that differ from the desired sequence.
- Can we speed up the process?

Subtask 3

Subtask 3 (15%): K = 1

- Yes, with partial sum / preprocessing
  - Count the number of 0/1 and compute their Partial Sum Arrays respectively
  - Assume the first 1 in resulting sequence starts at position p
  - Cost of achieving that sequence
    = (number of 1 from the start to p-1) + (number of 0 from p to the end)
  - Each query is asking for the Partial Sum of the number of 0/1 respectively
  - Both can be preprocessed in O(N) then accessed in O(1)

Score: 15 (Cumulative: 35)
Time Complexity: O(N)

Subtask 4

Subtask 4 (10%): N <= 200

- Borrow the idea from subtask 2:
  - Enumerate through all possible resulting sequences -> N + 1 possibilities
- Constraint of K = 1 is now gone
  - How to determine if a resulting sequence is achievable?

Subtask 4

How to determine if a resulting sequence is achievable?

Here we propose a strategy that might not sound intuitive:

- Let say we focus on one possible resulting sequence.
- Start from 1 to n, greedily perform operation whenever the current index's value differ from desired resulting sequence.
- This can either check if it's impossible or find the **only** solution the gives the resulting sequence.

Subtask 4

This can either check if it's impossible or find the **only** solution the gives the resulting sequence. → how??

- With the solution we found, we obtained the desire sequence.
- Assume there are solutions that differ from the one we found by this method.
  - i.e. there are positions that are / aren't performed operation on
- However, if such solutions exist and we look at the first differ position x:
  - Applying operation on x will make A[x] differ from the desired sequence.
  - By then, A[x] will never be correct since all operations afterwards doesn't cover x.

This may give some intuition on why the method works.

Subtask 4

Now we can try and go for the approach, for each desired sequence:

- For all O(N) possible resulting sequence:
- Start from 1 to N, perform operation whenever the current index's value differ from desired resulting sequence.
- Since a solution cannot contains the last K-1 positions, if there are elements in those positions are differ from the desired sequence after the previous operations, this sequence is impossible.
- Each operation takes O(K), and there are at most N operations in total.

Score: 10 (Cumulative: 45)
Time Complexity: $O(N^2K)$

Subtask 5

Subtask 5 (17%): N <= 5000

- Can we improve from subtask 4?
  - Each operation takes O(K) ← really?

Subtask 5

Notice how the operations are in form of range modification.

- Xor 1 on range(2, 5) is equivalent to Xor 1 on range(2, N) and then Xor 1 on range(6, N), because Xor 1 is performed twice on range(6, N) which they cancel out each other.
- We can store the modifications with a difference array
  - Denote `dif[]` as the difference array
  - Update e.g. on range(2, 5) can be transform into
    - `dif[2] ^= 1, dif[6] ^= 1`
  - The overall change on an index can then be accessed by xor-ing up all the values in `dif` before it.

Subtask 5

Back to solution for subtask 4, where for each desired sequence:

- For all O(N) possible resulting sequence:
- Start from 1 to N, perform operation whenever the current index's value differ from desired resulting sequence.
- Check is the resulting sequence possible.
- Each operation takes ~~O(K)~~ O(1), and there are at most N operations in total.

Score: 17 (Cumulative: 62)
Time Complexity: $O(N^2)$

Subtask 6

Subtask 6 (8%): K = 2

- The O(NK) solution we discussed just now can pass subtask 6.
- This task is made for people trying to improve from subtask 3.

Subtask 6

Subtask 6 (8%): K = 2

- Recall from subtask 3:
  - Cost of achieving a desired resulting sequence
    = (number of 1 before p) + (number of 0 after p−1)
- The idea can be brought to this subtask with slight modification and case handling.

Subtask 6

Instead of counting the number of 0 and 1, we have to instead count the number of operations applied to turn a certain range into 0 or 1.

- Besides, we may have to maintain extra information on the positions that are performed operations on so that we can make sure the desired sequence is achievable.
- Cost of achieving a desired resulting sequence

  = (number of ops from 1 to p-1) + (number of ops from p to N)

  (only if the operations from p - K to p - 1 matches with those after p - 1)
- For K = 2, this can be handled easily using several `if`.

Subtask 6

Both the solution would get us through subtask 6.

- Score: 8 (Cumulative: 70)
- Time Complexity: $O(NK)$ or $O(N)$

Partial solution

To move on, we can choose between two path:

1. Implement the O(N) solution on subtask 6
2. Seek for more observations

For path 1, the solution session ends here.

For path 2, let's take a look at the partial scoring:

Partial solution

Recall that: for all subtasks, you score 50% if your program can:

- detect invalid cases
- give a valid solution to valid cases with length not exceeding 10^6.

That being said, we do not have to look through all the possible N + 1 resulting sequences.
Can we only try 1 possible resulting sequences and get the solution working in O(N)?

Partial solution

But which resulting sequence should we try?

Perhaps try and fill as much 0 as possible?

Partial solution

But which resulting sequence should we try?

Perhaps try and fill as much 0 as possible? **Yes**

Partial solution

Perhaps try and fill as much 0 as possible? **Yes**

Assume a solution with x 1s at the end exists.
We can always achieve a solution with x % K 1s at the end.

- Just perform operation on each K indexes.

```
N = 40, K = 6
0000000000000000000011111111111111111    x = 19
                        ^     ^     ^
0000000000000000000000000000000000001    x = 1
```

Partial solution

A solution cannot contains the last K-1 positions

That means, if possible, a solution with less than K 1s must exists.

→ all indexes from 1 to N - K + 1 is 0

This can be found by filling as much 0 as possible.

- Score: 50
- Time Complexity: O(N)

Full solution

How does this lead to the full solution?

Given a possible resulting sequence, can we tell all the achievable sequences?

Full solution

Let's can take a bold guess:

- It is impossible to flip the values of a specified segments if its length is not a multiple of K.

You can verify using the strategy introduced in subtask 4:

- Start from 1 to n, perform operation whenever the current index's value differ from desired resulting sequence.

```
N = 20, K = 6
00000000000000000000     (try segment length of 4)
00000001111110000000
00000001111001111000
00000001111000000110     (impossible)
```

Full solution

Therefore, we now knows all the achievable resulting sequences, and they are easy to transfer to each other!

- The only possible way to achieve them is by performing / reverting operation for each K indexes.

This also allows us to calculate the number of operations needed easily.

```
N = 20, K = 6
00000000000000000011     (found possible)
00000000000011111111     (all other possible states)
00000011111111111111
01111111111111111111
```

Full solution

The final solutions goes like this:

1. Try and fill as much 0 as possible to check if it's possible.
2. If it's possible, store the positions of the operations in a set.
3. Transfer the solution into all other achievable solutions by adding/removing elements to/from the set (doing the same op twice will cancel out each other)
4. Check the operations needed for all solutions and pick the minimum one, or report if there's no solution.

- Score: 100
- Time Complexity: $O(N)$