EJOI 2021, Day 1,
English Editorial

August 26, 2021

Problem 1: Addk

(Proposed by Mihai Bunget.)

Subtask 1 A type 1 requirement is not processed because the array A does not change.

For query of type 2 we calculate the sum $S = A_l + A_{l+1} + ... + A_{l+m-1}$, which is added to the result, then for every i from $l + m$ to r $S = S - A_{i-m} + A_i$ is updated, which is added to the result.

Subtask 2 A type 1 requirement is not processed because the array A does not change.

For the type 2 requirement we observe the following two cases:

- If the length of the sequence A_l, A_{l+1}, ..., A_{r-1}, A_r, ie $r - l + 1$, is at least $2 \cdot m$, then the sum of the elements of all subsequences of length m in this sequence will be $S = S_1 + S_2 + S_3$, where $S_1 = 1 \cdot A_l + 2 \cdot A_{l+1} + ... + (m-1) \cdot A_{l+m-2}$, $S_2 = (r-l+1-2 \cdot (m-1)) \cdot (A_{l+m} + A_{l+m+1} + ... + A_{r-m+1})$, $S_3 = (m-1) \cdot A_{r-m+2} + ... + 2 \cdot A_{r-1} + 1 \cdot A_r$.
- If the length of the sequence A_l, A_{l+1}, ..., A_{r-1}, A_r is less than $2 \cdot m$, then the sum of the elements of all subsequences of length m in this sequence will be $S = S_1 + S_2 + S_3$, where $S_1 = 1 \cdot A_l + 2 \cdot A_{l+1} + ... + (rlm) \cdot A_{rm-1}$, $S_2 = (r-l+2-m) \cdot (A_{rm} + A_{r-m+1} + ... + A_{l+m-1})$, $S_3 = (rl-m+1) \cdot A_{l+m} + ... + 2 \cdot A_{r-1} + 1 \cdot A_r$.

For this we construct the arrays of partial sums B and C, so that $B_i = A_1 + A_2 + ... + A_i$ and $C_i = 1 \cdot A_1 + 2 \cdot A_2 + ... + i \cdot A_i$, for any i from 1 to N. These being constructed, for the first case above we will have $S_1 = C_{l+m-2} - C_{l-1} - (l-1) \cdot (B_{l+m-2} - B_{l-1})$, $S_2 = (r - l + 1 - 2 \cdot (m-1)) \cdot (B_{r-m+1} - B_{l+m-1}$, $S3 = (r+1) \cdot (Br - Br - m + 1) - (Cr - Cr - m + 1)$.

Subtask 3 Because in this case we also have the update operation on array A we will use the binary indexed tree structure for arrays B and C defined above.

Thus for the type 1 requirement we will update these BIT's (binary indexed trees), for each of the elements A_{i_1}, A_{i_2}, ..., A_{i_K} .

For the type 2 requirement we will use two interval calculation functions for arrays B and C, using the BIT's associated with these arrays. The calculation formulas are the same from subtask 2.

Problem 2: Kpart

(Proposed by Ionel-Vasile Piț-Rada.)

We will further note by $ksir(A, N, pos, kValues)$ the context of the solution for the string A_1, A_2, ..., A_N. The $kValues$ string contains, in ascending order, all the values K for which A is K-string. The pos string is defined by $pos[s]$= the highest index, between $1, 2, 3, ..., N$, for which there is a sub-sequence that starts with $A[pos[s]]$ and has the amount s. The values $pos[s]$ will be equal to -1 for the amounts s, $1 \leq s \leq 50000$, which have not yet been reached. The value $pos[0]$ is initially equal to 0.

Suppose we have solved the problem $ksir(A, N-1, pos, kValues)$ and we want to get the solution for the problem $ksir(A, N, pos, kValues)$. How can we proceed?

The newly appeared value is $A[N]$, so we will update in a first step the string pos by going through decreasing all the positive amounts s already reached and updating the $pos[s + A[N]]$ each time $s + A[N] \leq 50000$. The update will be made using the expression $pos[s + A[N]] = max(pos[s + A[N]], pos[s])$. Then it will update $pos[A[N]]$with N.

In the second stage we will go through the string $kValues$ and update it. Note that the values $kValues[j], 1 \leq j \leq length(kValues)$ are all already validated for the sub-string A_1, A_2, ..., A_{N-1}. What we are interested in now is to check/validate the sub-strings $A_{N-kValues[j]+1}...A_N$, checking if the amounts $s = A_{N-kValues[j]+1} + A_{N-kValues[j]+2} + ... + A_N$ are even and $pos[s/2] \geq N - kValues[j] + 1$, where $1 \leq j \leq length(kValues)$, which ensures that the amounts $s/2$ can be obtained using only elements of a sub-sequence inside the verified sub-string. The sub-string $A_1...A_N$ is checked/validated separately using the same idea. Obviously, all valid $kValues[j]$ values will be kept.

Initially it will start with the context of the $ksir(A, 0, pos, kValues)$ in which A is the empty string, $length(kValues) = 0$, $pos[0] = 0$ and $pos[j] = -1$ for $1 \leq j \leq 50000$, and then step by step the contexts will be updated $ksir(A, i, pos, kValues)$, for $1 \leq i \leq N$. The complexity is $O(T \cdot N \cdot S)$, where $S \leq 50000$.

Problem 3: Xcopy

(Proposed by Tulba-Lecu Theodor-Gabriel & Pop Ioan Cristian)

Disclaimer The subtasks were designed to give contestants a way to solve the task step by step. Each subtask adds a new level of difficulty and brings you closer to the optimal solution. The proofs for all the observations can be found at the end of the editorial in the Appendix section.

1 Solution for subtask 1

When we set $N = 1$, the task is reduced to finding an array such that any two consecutive numbers differ by exactly one bit. This is well known as a Gray code.

We can greedily generate a partial Gray code of length M that contains all the elements from 0 to $M-1$ exactly once. For the rest of the editorial we will refer to this as a "compact Gray code". A constructive algorithm can be found in the section A of the Appendix.

2 Solution for subtask 2

For this subtask we need to worry if and how the bits that change when moving on a row, influence those that change when moving on a column. It turns out that the two sets of bits are completely independent of each other (no bit can be influenced by both moving on a row or by moving on a column). A proof for this can be found in the section B of the Appendix.

Since both N and M are powers of 2, the Gray codes are complete and thus we only need to concatenate the Gray code for the rows, and columns. An example for such a test would be $N = 4$ and $M = 8$. A possible optimal answer would be:

0	1	3	2	6	7	5	4
8	9	11	10	14	15	13	12
24	25	27	26	30	31	29	28
16	17	19	18	22	23	21	20

3 Solution for subtask 3

The third subtask is a combination of the first two subtasks. You need to compute a "compact Gray code" for the column set of bits. And then concatenate the bits of the two sets in a way to allow an optimal answer.

4 Complete solution

The last observation we need to make in order to solve the problem is how to combine the two sets of bits to obtain a minimal value. Just concatenating doesn't necessarily produce an optimal outcome. For example for $N, M = 6$ if we generate the following "compact Gray code" for both sets of bits:

$$5 = 101_{(2)}, 1 = 001_{(2)}, 3 = 011_{(2)}, 2 = 010_{(2)}, 0 = 000_{(2)}, 4 = 100_{(2)}$$

and concatenate the answers we would obtain the following matrix:

101101	101001	101011	101010	101000	101100
001101	001001	001011	001010	001000	001100
011101	011001	011011	011010	011000	011100
010101	010001	010011	010010	010000	010100
000101	000001	000011	000010	000000	000100
001101	001001	001011	001010	001000	001100

with a maximum value of $45 = 101101$. Whereas if we look at the maximum value ($5 = 101_{(2)}$), we can observe that we can merge the two 5s in the following way: $43 = 101011$. Thus we can minimize the maximum value of the matrix to 43 instead of 45 and obtain the following matrix:

101011	100011	100111	100110	100010	101010
001011	000011	000111	000110	000010	001010
011011	010011	010111	010110	010010	011010
011001	010001	010101	010100	010000	011000
001001	000001	000101	000100	000000	001000
101001	100001	100101	100100	100000	101000

The merging can be done in various ways, one of which would be using dynamic programming:

$dp_{i,j}$ = the minimum value that can be obtained using the first i bits of the maximum of the first set and the first j bits of the maximum of the second set.

A Compact Gray code constructive algorithm

Gray codes of power-of-two length. Note that a Gray code of length 2^n can be created by setting element i (indexed from 0) to $i \oplus (i \texttt{ div } 2)$. This Gray code can also be rotated (thus if $g_1, \ldots, g_{2^n}$ is a Gray code, so is $g_k, \ldots, g_{2^n}, g_1, \ldots, g_{k-1}$. We leave as an exercise to the reader proving that this works.

Gray codes of arbitrary length. Observe the following facts:

- We can create a Gray code of length 2^n starting with any number from 0 to $2^n - 1$, using the previous construction suitably rotated.
- Any integer is a sum of distinct powers of 2.

These facts suggest immediately an algorithm: create Gray codes of lengths equal to the powers of 2 that compose the length of the desired Gray code, and put them together in some way. We will show how this is done by example, for length 7.

Note that Gray codes for length $1, 2$ and 4 respectively are $0, 1, 3, 2$, $0, 1$ and 0 respectively. We now "add in" the high order bit in the second and third Gray codes to get $0, 1, 3, 2$, $4, 5$, 6. Now we need to rotate the Gray codes so that they can be adjacent, from right to left. 6 can only be adjacent to 4, so the array ends with $5, 4, 6$. 5 can only be adjacent to 1, so the full array is $3, 2, 0, 1, 5, 4, 6$.

B The row-column independence

First let's note that if we fix the elements at the indexes (i, j), $(i + 1, j)$ and $(i, j + 1)$, then the element at the index $(i + 1, j + 1)$ is uniquely determined by the following formula ($\oplus$ represents the bitwise XOR operation):

$$M[i + 1][j + 1] = M[i][j + 1] \oplus M[i + 1][j] \oplus M[i][j]$$

More than that, we can recursively extend this formula to:

$$M[i][j] = M[0][j] \oplus M[i][0] \oplus M[0][0]$$

Now let's suppose that there exists a bit such that it is changed both on a row and a column. That means that there exists a bit k and a row i and column j such that:

- $M[i + 1][x] = M[i][x] \oplus 2^k, \forall\ x \in \{0, 1, \ldots, M - 1\}$
- $M[x][j + 1] = M[x][j] \oplus 2^k, \forall\ x \in \{0, 1, \ldots, N - 1\}$

From the previous relation we get the following result:

$$M[i+1][j+1] = M[i][j+1] \oplus M[i+1][j] \oplus M[i][j]$$

$$M[i+1][j+1] = M[i][j] \oplus 2^k \oplus M[i][j] \oplus 2^k \oplus M[i][j]$$

$$M[i+1][j+1] = M[i][j]$$

In conclusion, if there would exist a bit that is influenced both by a line and by a column, then the condition of distinctness would not be satisfied.

Scientific committee

The problems were prepared by:

- Adrian Panaete (chair) - "A.T. Laurian" National College, Botoșani
- Ionel-Vasile Piț-Rada - "Traian" National College, Drobeta Turnu Severin
- Maria-Alexa Tudose - University of Oxford, UK
- Gheorghe-Eugen Nodea - "Tudor Vladimirescu" National College, Târgu Jiu
- Tamio-Vesa Nakajima - Oxford, Computer Science department, UK
- Mihai Bunget - "Tudor Vladimirescu" National College, Târgu Jiu
- Andrei-Costin Constantinescu - University of Oxford, UK
- Ciprian-Daniel Chește - "Grigore C. Moisil" Technological High School, Buzău
- Lucian Bicsi - University of Bucharest
- Dan Pracsiu - "Emil Racoviță", Theoretical High School, Vaslui
- Ioan-Cristian Pop - Polytechnical University, Bucharest
- Theodor-Gabriel Tulbă-Lecu - Polytechnical University, Bucharest
- Raluca-Veronica Costineanu - "Ștefan cel Mare" National College, Suceava