Zonal Computing Olympiad 2024
Editorial and Results

Indian Computing Olympiad Scientific Committee

April 7, 2024

1 Vegetables

Author: Soumyaditya Choudhuri

Preparation: Aryan Maskara, Paras Kasmalkar, Shreyan Ray

Abridged statement: There are two arrays of positive integers, A and B, both of length N. In one operation, you can select any positive element from either array and decrement it by one. The score is the value of $\sum_{i=1}^{N} A_i \times B_i$ after all operations. What is the minimum possible score you can achieve after applying at most X_j operations? Find the answer for Q different values of X_j.

Constraints:

- $1 \leq N, Q \leq 2 \cdot 10^5$.
- $1 \leq A_i, B_i \leq 10^6$ for all $1 \leq i \leq N$.
- $1 \leq X_j \leq 10^9$ for all $1 \leq j \leq Q$.

Subtasks:

- **Subtask 1** (7 points) $Q = 1$, $X_j = 1$, $N \leq 10$, $A_i \leq 10^3$, $B_i \leq 10^3$.
- **Subtask 2** (6 points) $Q \leq 3$, $X_j \leq 3$, $N \leq 10$.
- **Subtask 3** (9 points) $A_i = 1$, $N \leq 10^3$, $Q \leq 10^3$.
- **Subtask 4** (7 points) $A_i = 1$.
- **Subtask 5** (18 points) $N, Q, X_j \leq 30$.
- **Subtask 6** (15 points) $N, Q, X_j \leq 400$.
- **Subtask 7** (16 points) $N, Q \leq 10^3$.

- **Subtask 8** (7 points) The sum of all A_i does not exceed $2 \cdot 10^5$. Also, the sum of all B_i does not exceed $2 \cdot 10^5$.
- **Subtask 9** (7 points) $X_j \leq 2 \cdot 10^5$.
- **Subtask 10** (8 points) No additional constraints.

1.1 Subtask 1: (7 points) $Q = 1$, $X_j = 1$, $N \leq 10$, $A_i \leq 10^3$, $B_i \leq 10^3$

In this subtask, we are only allowed one operation. There are $2 \cdot N$ possibilities for this operation, and we can try all of them and compute the final score in $O(N)$ time for each one. The time complexity is $O(N \cdot (N + M))$.

1.2 Subtask 2: (6 points) $Q \leq 3$, $X_j \leq 3$, $N \leq 10$

In this subtask, we can brute force using recursion to try all possibilities of operations. We can try every sequence of operations. There are at most $2N$ different operations in each step, and there are at most three allowed operations. Therefore, the time complexity is $O(N^3)$ per query for a total of $O(QN^3)$.

1.3 Subtask 3: (9 points) $A_i = 1$, $N \leq 10^3$, $Q \leq 10^3$

In this subtask, all A_i values are 1. Observe that for any index i, we can use a single operation on A_i and reduce the contribution of index i to 0. This causes a reduction in the total score of B_i. Also, notice that with N or more moves, it is possible to reduce the total score to 0.

Notice that it is always optimal to reduce the A_i value for which the corresponding B_i value is maximum. Therefore, we can sort the array of B values in descending order, and we should take the first X_j values to decrease by. This sorting will take $O(N \log N)$ time.

For each query, we can find the largest X_j values of B_i and compute their total reduction in at most $O(N)$ time.

The time complexity is $O(Q \cdot N + N \log N)$.

1.4 Subtask 4: (7 points) $A_i = 1$

This subtask requires an optimized version of the solution to Subtask 3. We can take the prefix sums of the sorted B array in the previous subtask. This allows us to answer any query in constant time.

The time complexity is $O(N \log N + Q)$.

1.5 Subtask 5: (18 points) $N, Q, X_j \leq 30$

Notice that for each term, only the number of moves made on each of the A_i and the B_i terms matter.

We can use dynamic programming to solve this subtask. Let `dp[i][u]` represent the minimum sum of the first `i` terms if `u` upgrades have been used so far.

The base case is that the sum of the first 0 terms is 0, regardless of the number of upgrades performed. That is, `dp[0][u] = 0` for all `u`.
For the transition, we can iterate over x, the number of upgrades performed on A, and y, the number of upgrades performed on B. Then, the total number of moves made is $x + y$. We must ensure that `A[i] >= x` and `B[i] >= y` before trying this move, as well as `u >= x + y`.

`dp[i][u]` $= \max\{$`dp[i-1][u - x - y] + (A[i] - x) * (B[i] - y)` $|$ all valid `x`, `y`$\}$

There are $O(X \cdot X)$ transitions to consider from each state, and a total of $O(N \cdot X)$ states. Therefore, the total complexity per query is $O(NX^3)$, for a total complexity of $O(QNX^3)$. This can be trivially optimised to $O(NX^3)$ by not recomputing the DP table every query, given that all answers can be precomputed. However, this optimisation is not necessary to solve this subtask.

1.6 Subtask 6: (15 points) $N, Q, X_j \leq 400$

For a given index i with values A_i and B_i (assume without loss of generality that $A_i \leq B_i$), consider the two following cases:

- Case 1: Make x moves on A_i, make y moves on B_i.
- Case 2: Make $x + y$ moves on A_i.

Note that it may be possible to perform case 1 for a given x and y but impossible to perform case 2. We will temporarily ignore the restriction that A and B must be non-negative in order to demonstrate the result.
The final contribution of the term in each case is:

- Case 1: $(A_i - x) \cdot (B_i - y) = A_i \cdot B_i - y \cdot A_i - x \cdot B_i + xy$ (denote this by d_1)
- Case 2: $(A_i - x - y) \cdot B_i = A_i \cdot B_i - B_i(x + y)$ (denote this by d_2)

We claim that $d_2 \leq d_1$.
This is not hard to see as $d_1 - d_2 = xy + y(B_i - A_i)$, which is non-negative since $B_i \geq A_i$ and x and y must both be non-negative. Note that when $x + y$ moves cannot be used on A_i, it means it get reduced to 0 beforehand, and thus there is no point doing any further moves on this index.
Hence, its optimal to use all moves on either A_i or B_i (whichever is smaller). This optimizes our dynamic programming to only need $O(X)$ transitions, thus leading to a $O(NX^2)$ solution. Note that unlike the previous subtask, you will need to precalculate all the values beforehand.

1.7 Subtask 7: (16 points) $N, Q \leq 10^3$

Note that from the previous subtasks, we only apply the moves to the minimum among A_i and B_i. Again, without loss of generality, assume $A_i \leq B_i$.

Let us consider the reduction in the term $A_i \cdot B_i$ when we apply one operation on A_i. The new value is $(A_i - 1) \cdot B_i$, which is smaller by B_i. Thus, if we only had one upgrade possible, then its optimal to do it on the index with maximum value of B_i (again assuming $A_i \leq B_i$).

This leads us to an greedy algorithm : Use moves on the maximum value of B_i until you exhaust it (i.e. when the corresponding A_i goes to 0). We can prove that this greedy is correct via Exchange Arguments.

For this subtask, the only thing left to do is sort indexes according to B_i (descending order) and use as many upgrades as possible on the first index, then the second index (if there are still upgrades left) and then the third and so on.

Time Complexity is $O(N \cdot Q)$

1.8 Subtask 8: (7 points) The sum of all A_i does not exceed $2 \cdot 10^5$. Also, the sum of all B_i does not exceed $2 \cdot 10^5$

Let us try to optimize our subtask 7 solution. Note that since the sum of A values and the sum of B values are both bounded by $2 \cdot 10^5$, after at most $2 \cdot 10^5$ upgrades, we can get the water requirement to 0.

Thus, we only need to simulate all upgrades, and store the answers for all of them. We can perform the greedy algorithm once, and at each step store the sum of $A_i \cdot B_i$.

When answering queries, we simply check if the number of upgrades allowed is greater than the number of simulated steps; if it is, then the answer is 0. Otherwise, we can read the value from our array of simulated results. Then the complexity is $O(S + Nlog(N) + Q)$, where S is bounded by $2 \cdot 10^5$.

1.9 Subtask 9: (7 points) $X_j \leq 2 \cdot 10^5$

The solution is the same as subtask 8, but the total number of upgrades possible might exceed $2 \cdot 10^5$. However, since only the first $2 \cdot 10^5$ upgrades are queried about, it is sufficient to simply simulate the first $2 \cdot 10^5$ upgrades.

Complexity is $O(\max\{X_j\} + N \log N + Q)$, and note that $\max(X_j)$ is bounded by $2 \cdot 10^5$ for this subtask.

1.10 Subtask 10: (8 points) No additional constraints

Let C_i denote the answer for a query with i updates. Note that the value of $C_{i+1} - C_i$ changes at a maximum of N points (i.e. when an index gets exhausted and we move on to the next one).

First, consider $A_i \leq B_i$ for all indices, and further the indices are sorted by A_i. Then D_i denote $\sum_{j=1}^{i} B_j$. Then, for $x = D_i$, the value of $C_{x+1} - C_x$ changes from B_i to B_{i+1}.

Let us store the values of D_i and the corresponding the sums of the array. Now suppose k is the last index such that $D_k \leq X_j$ and S_k is the corresponding memoized sum. Then, the answer is simply $S_k - (X_j - D_k) \cdot B_{k+1}$ because from D_k onwards, the value of $C_{i+1} - C_i$ is B_{k+1} and it only changes again at D_{k+1} (which we haven't reached yet, since we defined k to be the last index).

To implement this, you can use a map to store the memoized results, and binary search to find the relevant value of k. Alternatively, you can store two parallel prefix sum arrays and binary search on the arrays.

The time complexity is $O(N \log N + Q \log N)$.

2 Fruits

Author: Soumyaditya Choudhuri

Preparation: Aryan Maskara, Paras Kasmalkar, Shreyan Ray

Abridged statement: There is a N by M grid of integers, where each integer is from 1 to K. The distance between two cells on the grid is defined as the Manhattan distance between the two cells. Handle Q queries of the following two types: Query type 1 updates the value of a cell in the grid to some new value. Query type 2 gives you two distinct integers between 1 and K, call them U and V, and asks you to find the maximum distance between a cell with number U and a cell with number V.

Constraints:

- $1 \leq N, M \leq 3 \cdot 10^5$.
- $2 \leq K \leq N \cdot M \leq 3 \cdot 10^5$.
- $1 \leq Q \leq 2 \cdot 10^5$.

Subtasks:

- **Subtask 1** (7 points) : $Q = 1, N \cdot M \leq 100$
- **Subtask 2** (8 points) : $Q = 1, N \cdot M \leq 500$.
- **Subtask 3** (13 points) : $Q = 1, N \cdot M \leq 2000$.
- **Subtask 4** (18 points) : $Q = 1$.
- **Subtask 5** (12 points) : $N = 1$ and there will be no operations of type 1.
- **Subtask 6** (9 points) : $N = 1$.
- **Subtask 7** (13 points) : $N \leq 5$ and there will be no operations of type 1.
- **Subtask 8** (12 points) : There will be no operations of type 1.
- **Subtask 9** (8 points) : No additional constraints.

2.1 Subtask 1 (7 points) : $Q = 1, N \cdot M \leq 100$

This subtask is intended to reward various less efficient versions of the solutions to future subtasks. One example is using a Dijkstra in place of a BFS in Subtask 2.

2.2 Subtask 2 (8 points) : $Q = 1, N \cdot M \leq 500$.

Consider a graph where the nodes are the fields, and there is an edge between every pair of adjacent fields. There are potentially $O(NM)$ fields that grow fruit U, and $O(NM)$ fields that grow fruit V, for a total of $O((NM)^2)$ pairs. For each such pair, we can find the shortest path between them in $O(NM)$ using a BFS.

The time complexity is $O((NM)^3)$.

2.3 Subtask 3 (13 points) : $Q = 1, N \cdot M \leq 2000$.

Because the graph is a grid, the distance between any two cells (x_1, y_1) and (x_2, y_2) is the Manhattan distance between them, $|x_1 - x_2| + |y_1 - y_2|$. This can be computed in $O(1)$.

We can check the Manhattan distance between all possible pairs of fields.

The time complexity is $O((NM)^2)$.

2.4 Subtask 4 (18 points) : $Q = 1$.

2.4.1 Solution 1

For each field with fruit U, we can find the farthest field with fruit V in each quadrant individually (that is, top-left, top-right, bottom-left and bottom-right), and then take the largest distance among these.

Here, we describe how to find the farthest field in the top-left quadrant. We can use an identical algorithm in different directions to find the farthest fields in the other quadrants.

Consider a field (x_1, y_1) with fruit U. For all fields (x_2, y_2) in the top-left quadrant, we have $x_2 \leq x_1$ and $y_2 \leq y_1$. So, the Manhattan distance between the two fields is $(x_1 - x_2) + (y_1 - y_2) = (x_1 + y_1) - (x_2 + y_2)$.

In order to find the farthest field in the top-left quadrant, we need to find the field with the smallest possible value of $x_2 + y_2$. So, the problem reduces to finding the smallest value of $x_2 + y_2$ among fields growing fruit V for the top-left quadrant of every field.

This can be done using dynamic programming. Let `dp[i][j]` be the smallest value of $x_2 + y_2$ among fields growing fruit V such that $1 \leq x_2 \leq i$ and $1 \leq y_2 \leq j$ (or infinity, if such a field does not exist). We can compute this DP using the following recurrence:

$$\texttt{dp[i][j]} = \begin{cases} \texttt{i + j}, & \text{if } \texttt{dp[i-1][j] == } \infty \text{ and } \texttt{dp[i][j-1] == } \infty \\ \min(\texttt{dp[i-1][j]}, \texttt{dp[i][j-1]}), & \text{otherwise} \end{cases}$$

The time complexity is $O(NM)$.

2.4.2 Solution 2

An alternative solution to this subtask involves simply finding, for each combination of fruit (from 1 to K) and each corner (top-left, bottom-left, top-right, bottom-right), the closest fruit of each type to each corner. Then, this reduces the candidates to just four cells per fruit (one for each corner). It is sufficient to check only these candidates, so a total of $4 \times 4 = 16$ pairs can be searched. Finding these 4 items for each fruit can be done in $O(N \cdot M)$ time, and the overall complexity of this solution is also $O(N \cdot M)$.

2.5 Subtask 5 (12 points) : $N = 1$ and there will be no operations of type 1.

This subtask requires an important observation. Consider some two fruits U and V. In the optimal pair of cells, let the field growing fruit U be $(1, a)$ and the field growing fruit V be $(1, b)$. The distance between a and b is $|a - b|$.

Without loss of generality, assume that $i < j$. If there is some cell smaller than a that grows fruit U, then it is optimal to switch a to that cell, so the current pair (a, b) is not optimal. This implies that a must be the leftmost field growing fruit U. Similarly, b must be the rightmost cell growing fruit V.

For each fruit, only the leftmost and rightmost cells are relevant. We can compute these cells for each color in the beginning. While answering a query, we can check $O(1)$ pairs of cells.

The time complexity is $O(M + Q)$.

2.6 Subtask 6 (9 points) : $N = 1$.

The solution is the same as subtask 5, but we can maintain the set of fields containing each fruit in a balanced binary search tree such as the C++ `std::set` or `std::multiset` (depending on implementation), which enables efficient updates.

The time complexity is $O(M \log M + Q \log M)$.

2.7 Subtask 7 (13 points) : $N \leq 5$ and there will be no operations of type 1.

We can use a similar solution to subtask 5, except we need to store the leftmost and rightmost locations of each fruit for each row and also consider all pairs of rows when trying to find the answer.

The time complexity is $O(N^2(M + Q))$.

2.8 Subtask 8 (12 points) : There will be no operations of type 1.

We extend the idea from subtask 5 to two dimensions.

The absolute value of any number z is $|z| = \max(z, -z)$. Consider two cells (x_1, y_1) and (x_2, y_2). The Manhattan distance between them, $|x_1 - x_2| + |y_1 - y_2|$, can be written in another form:

$|x_1 - x_2| + |y_1 - y_2| = \max(x_1 - x_2, x_2 - x_1) + \max(y_1 - y_2, y_2 - y_1)$

$= \max((x_1 - x_2 + y_1 - y_2),\ (x_1 - x_2 + y_2 - y_1),\ (x_2 - x_1 + y_1 - y_2),\ (x_2 - x_1 + y_2 - y_1))$

$= \max((x_1+y_1)-(x_2+y_2), (x_1-y_1)-(x_2-y_2), (x_2-y_2)-(x_1-y_1), (x_2+y_2)-(x_1+y_1))$

$= \max(\max((x_1 + y_1) - (x_2 + y_2), (x_2 + y_2) - (x_1 + y_1)), \max((x_1 - y_1) - (x_2 - y_2), (x_2 - y_2) - (x_1 - y_1)))$

$= \max(|(x_1 + y_1) - (x_2 + y_2)|, |(x_1 - y_1) - (x_2 - y_2)|)$

We can simply find the maximum possible value of $|(x_1 + y_1) - (x_2 + y_2)|$ among all pairs of cells and the maximum possible value of $|(x_1 - y_1) - (x_2 - y_2)|$ among all pairs of cells, and take the maximum of both of these to find the answer.

The problem of finding the maximum possible value of $|(x_1 + y_1) - (x_2 + y_2)|$ is very similar to subtask 5, and we use the same idea of precomputing the minimum and maximum $x + y$ values for all fruits. We need to do a similar precomputation for $|(x_1 - y_1) - (x_2 - y_2)|$. It is also possible to derive this solution via a precomputation of Solution 2 to subtask 4.

The time complexity is $O(NM + Q)$.

2.9 Subtask 9 (8 points) : No additional constraints.

Similar to Subtask 6, we can use sets to enable efficient updates in the solution of Subtask 8.

The time complexity is $O(NM \log(NM) + Q \log(NM))$.

3 Scientific Committee

The problem setting committee consisted of, in alphabetical order:

- Aryan Maskara, International Institute of Information Technology, Hyderabad
- Jishnu Roychoudhury, Princeton University
- Paras Kasmalkar, University of Wisconsin-Madison
- Shreyan Ray, IIT Kharagpur
- Soumyaditya Choudhuri, Carnegie Mellon University

4 Results and Statistics

A total of 313 contestants participated in this contest, obtaining a total of 51 distinct scores out of 200 possible points. This section features the breakdown of results by problem, subtask, and over the whole contest.

4.1 Vegetables: Problem Statistics

This problem was scored out of a possible 100 points. The statistics are given below:

- Maximum score: 100
- Minimum score: 0
- Mean score: 28.6
- Median score: 13
- Distinct scores: 19

The following table summarises the number of participants who solved each subtask of this problem:

Subtask	Participants solving this subtask
1	218
2	145
3	102
4	67
5	117
6	105
7	47
8	36
9	37
10	30

The following table summarises the score distribution on this problem:

Score	Contestants at this score	Contestants at or above this score	Ranks
100	30	30	1-30
92	1	31	31
85	1	32	32
78	8	40	33-40
76	3	43	41-43
71	7	50	44-50
62	15	65	51-65
55	9	74	66-74
47	1	75	75

46	31	106	76-106
40	3	109	107-109
31	8	117	110-117
29	6	123	118-123
23	6	129	124-129
22	5	134	130-134
16	7	141	135-141
13	17	158	142-158
7	60	218	159-218
0	95	313	219-313

4.2 Fruits: Problem Statistics

This problem was scored out of a possible 100 points. The statistics are given below:

- Maximum score: 100
- Minimum score: 0
- Mean score: 15.9
- Median score: 0
- Distinct scores: 15

The following table summarises the number of participants who solved each subtask of this problem:

Subtask	Participants solving this subtask
1	119
2	117
3	117
4	16
5	53
6	25
7	22
8	13
9	11

The following table summarises the score distribution on this problem:

Score	Contestants at this score	Contestants at or above this score	Ranks
100	11	11	1-11
92	1	12	12
83	1	13	13
80	1	14	14
71	1	15	15

62	5	20	16-20
53	2	22	21-22
49	7	29	23-29
46	1	30	30
40	20	50	31-50
28	67	117	51-117
19	1	118	118
12	3	121	119-121
7	1	122	122
0	191	313	123-313

4.3 Overall Score Distribution

Score	Contestants at this score	Contestants at or above this score	Ranks
200	10	10	1-10
192	1	11	11
172	1	12	12
162	5	17	13-17
149	3	20	18-20
146	1	21	21
145	1	22	22
140	6	28	23-28
129	2	30	29-30
128	3	33	31-33
127	1	34	34
124	1	35	35
120	1	36	36
118	5	41	37-41
116	1	42	42
106	1	43	43
104	2	45	44-45
102	3	48	46-48
100	1	49	49
99	3	52	50-52
94	1	53	53
90	10	63	54-63
83	5	68	64-68
80	1	69	69
74	9	78	70-78
71	1	79	79
69	2	81	80-81
68	2	83	82-83
63	2	85	84-85
62	3	88	86-88

59	2	90	89-90
57	3	93	91-93
56	1	94	94
55	5	99	95-99
51	1	100	100
50	2	102	101-102
47	1	103	103
46	22	125	104-125
44	3	128	126-128
41	7	135	129-135
40	1	136	136
35	7	143	137-143
31	5	148	144-148
28	10	158	149-158
23	3	161	159-161
22	2	163	162-163
19	1	164	164
16	3	167	165-167
13	10	177	168-177
7	51	228	178-228
0	85	313	229-313

5 Cutoffs for qualification to INOI 2024

Grade	Male	Female
12	55	35
11	51	35
10	50	35
9	46	28
8	41	28
7 and below	41	28