Data Structure and Algorithm, Spring 2022

Homework 2

Due: 13:00:00, Tuesday, April 26, 2022

TA E-mail: dsa_ta@csie.ntu.edu.tw

Rules and Instructions

- Any form of cheating, lying, or plagiarism will not be tolerated. Students can get zero scores and/or fail the class and/or be kicked out of school and/or receive other punishments for those kinds of misconducts.
- In Homework 2, the problem set contains a special Problem 0 (see below) and 5 other problems. The set is divided into two parts, the non-programming part (Problems 0, 1, 2, 3) and the programming part (Problems 4, 5).
- For problems in the non-programming part, you should combine your solutions in ONE PDF file. Your file should generally be legible with a white/light background—using white/light texts on a dark/black background is prohibited. Your solution must be as simple as possible. At the TAs' discretion, solutions which are too complicated can be penalized or even regarded as incorrect. If you would like to use any theorem which is not mentioned in the classes, please include its proof in your solution.
- The PDF file for the non-programming part should be submitted to Gradescope as instructed, and you should use Gradescope to tag the pages that correspond to each sub-problem to facilitate the TAs' grading. Failure to tagging the correct pages of the sub-problem can cause a 20% penalty on the sub-problem.
- For the programming part, you should have visited the *DSA Judge* (https://dsa-2022.csie.org) and familiarized yourself with how to submit your code via the judge system in Homework 0.
- For problems in the programming part, you should write your code in C programming language, and then submit the code via the judge system. In each day, you can submit up to 5 times per day for each problem. To encourage you to start early, we allow 10 times of submissions per day in the first week (2022/03/29-2022/04/04). The judge system will compile your code with

```
gcc main.c -static -O2 -std=c11
```

- Discussions on course materials and homework solutions are encouraged. But you should write the final solutions alone and understand them fully. Books, notes, and Internet resources can be consulted, but not copied from.
- Since everyone needs to write the final solutions alone, there is absolutely no need to lend your homework solutions and/or source codes to your classmates at any time. In order to maximize the level of fairness in this class, lending and borrowing homework solutions are both regarded as dishonest behaviors and will be punished according to the honesty policy.
- The score of the part that is submitted after the deadline will get some penalties according to the following rule:

$$\text{LateScore} = \max\left(0, \frac{86400 \cdot 5 - \text{DelayTime (sec.)}}{86400 \cdot 5}\right) \times \text{OriginalScore}$$

- If you have questions about HW2, please go to the discord channel and discuss (*strongly preferred*, which will provide everyone a better learning experience). If you really need an email answer, please follow the rules outlined below to get a fast response:
  - The subject should contain two tags, `"[hw2]"` and `"[Px]"`, specifying the problem where you have questions. For example, `"[hw2][P5] Is k in sub-problem 3 an integer"`. Adding these tags allows the TAs to track the status of each email and to provide faster responses to you.
  - If you want to provide your code segments to us as part of your question, please upload it to Gist or similar platforms and provide the link. Please remember to protect your uploaded materials with proper permission settings. Screenshots or code segments directly included in the email is discouraged and may not be reviewed.

Non-Programming Part

Problem 0 - Proper References (0 pts)

For each problem below, please specify the references (the Internet URL you consulted with or the classmates/friends you discussed with; you do not need to list the TAs/instructors) in your PDF submitted to Gradescope. If you finished any problem all by yourself (with the help of TAs/instructors), just say "all by myself." While we did not allocate any points on this problem, failure to complete this problem can lead to penalty (i.e. negative points). Examples are

- Problem 1: Alice, Bob, and https://stackoverflow.com/questions/982388/
- Problem 2: all by myself
- ...

Listing the references in this problem does *not* mean you could copy from them. We cannot stress this enough: *you should always write the final solutions alone and understand them fully.*

Problem 1 - Lucy's Laptop (100 pts)

When studying in the basement of the De-Tian building, you must be very careful if you want to leave your seat and go to the restroom. In particular, if you forget to lock your laptop before leaving, some kind friends may choose to remind you to lock your screen with some prank—in computer science terminology, help you increase the security level with an adversarial attack. Here is how the prank generally works when a student goes to the restroom with his laptop open.

1. Open `https://www.facebook.com` in the web browser, which should most certainly be logged in.
2. Click **Create Post** to create a surprising post such as giving out a free RTX3070 graphics card.
3. Click **Post**, turn off the notification of the post, refresh the page (to set the post to read), and close the browser, all before the student gets back.
4. Other friends who see the post might reply "抽", which reminds the student to lock the laptop next time.

Lucy, a senior student of CSIE, is a careless person and thus has a very colorful Facebook main page. But that did not change Lucy's carelessness. Therefore, Lucy's friend decides to try some different pranks. One day, Lucy came back from the restroom and discovered some challenge problems under a suspicious folder. Please help Lucy solve those.

1. (15 pts) Given the inorder and postorder traversals of a binary tree as follows.

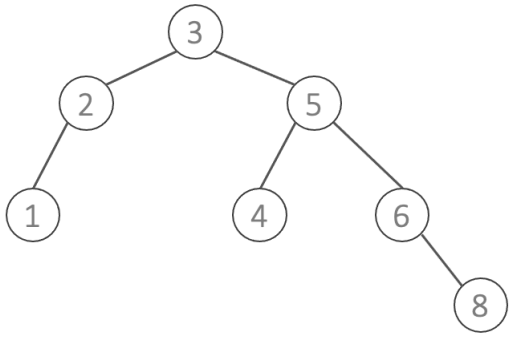
   ```
   inorder traversal: 4, 26, 19, 22, 7, 15, 34, 11, 13, 8
   postorder traversal: 26, 19, 4, 7, 34, 15, 8, 13, 11, 22
   ```

   Please reconstruct the original binary tree and draw it. Briefly explain the "human algorithm" that you use to reconstruct the original binary tree.

2. (15 pts) The prankster further challenges Lucy in modifying a binary search tree T with tree nodes with unique values $\{a_i\}$ to another binary tree T' with the same connections and new nodes $\{b_i\}$ such that

$$b_i = a_i + \sum_{v \in S_i} v.val \quad , S_i = \{v | v \in T, v.val > a_i.val\}$$

That is, b_i is the sum of the value of every node in T that is of value $\geq a_i$. Note that T' might not be a binary search tree. Now given a binary search tree T below, please draw the modified tree T'. No more explanations are needed—we only need the resulting T'.



3. (15 pts) Given a binary search tree T with root pointer *root*. Design an algorithm (with pseudo code, or even more preferably, with understandable words) that modifies T to T' for the previous sub-problem in $O(n)$ time, where n is the number of nodes in T, and briefly explain why the algorithm can modify T to T' correctly.

4. (20pts) Prove or disprove the following statement, using only the definition of the binary search tree.

   "For a binary search tree with no repeated value, let x be a leaf node and y be its parent. Then y is either the smallest node among all nodes larger than x, or the largest node among all nodes smaller than x."

5. (15pts) Construct a complete binary search tree from the integer sequence below and draw the tree. No more explanations are needed—we only need the resulting tree.

   `{58, 71, 14, 35, 26, 32, 74, 83, 59, 3}`

6. (20pts) In our class, we discussed how a complete binary tree can be easily stored in an array a with the following recursive definition:

   - Store the root in $a[1]$.
   - For every node located at $a[i]$, store its left child in $a[2i]$ and its right child in $a[2i+1]$.

   The same definition can be used to store an arbitrary binary tree with some space waste on the "incomplete" nodes. That is, some elements $a[i]$ would not hold any valid values in the original binary tree. For instance, for the binary tree in sub-problem 2, the array representation wastes 8 positions within the array:

   $$a = \texttt{\{3, 2, 5, 1, x, 4, 6, x, x, x, x, x, x, x, 8\}}$$

   with `x` denoting the wastes space.

   Now, please design an algorithm that calculates the number of wasted positions when storing a binary tree T within an array. The algorithm should run within $O(n)$ time, where n is the number of nodes in T. But you *cannot* actually construct the array. Explain the algorithm (with pseudo code, or even more preferably, with understandable words) and why it meets the required time complexity.

Problem 2 - Teaching Assistant to Music Teacher (100 pts + 20 Bonus pts)

Suppose you are a teaching assistant to your music teacher in the senior high school. Your responsibility is to arrange the order of the presentation groups "properly" to prevent your music teacher from getting angry. There are n presentation groups in your class with $n \geq 3$. For your music teacher, each group i has an "attitude value" a_i that is closely related to how mad your music teacher will get after the group's presentation. You do not know the exact values of $a_1, a_2, \cdots, a_n$, but you can ask your teacher some questions (carefully) to obtain some relation between those values. The teacher's answer to your query can be represented by the following black-box function:

- `query_attitude_value(i,j,k)`: The parameters are three **distinct** indices $1 \leq i, j, k \leq n$. If $a_i \leq a_j \leq a_k$ or $a_i \geq a_j \geq a_k$, then this function will return TRUE; otherwise, it will return FALSE.

For all the sub-problems below, we will focus on the "query complexity" of your algorithm, where the query complexity is on the number of calls to the black-box function described above. That is, we assume that all the implementation details, such as control, data movement, etc. does not take any efforts. We just focus on whether we have the necessary information from calling the `query_attitude_value` function to complete the task.

1. (20 pts) Given the black-box function, design an algorithm with $O(n)$ query complexity to return the boundary groups $\{a_m, a_M\}$, where a_m is of the smallest attitude value and a_M is of the largest attitude value. You do not have to (and perhaps cannot) distinguish a_m and a_M. You just need to return both of them. Illustrate your algorithm with pseudo code, or even more preferably, with understandable words, and briefly explain why it runs within the demanded query complexity.

2. (20 pts) Using the $O(n)$ algorithm in the sub-problem above, design an algorithm that sorts all presentation groups such that their attitude values are monotonic (i.e. either non-increasing or non-decreasing). Your algorithm needs to run in $O(n \log n)$ query complexity in the worst case or in the average case (with respect to a uniform distribution over all possible permutations of attitude values). Illustrate your algorithm with pseudo code, or even more preferably, with understandable words, and briefly explain why it runs within the demanded query complexity. You can use the result in the previous sub-problem if needed, even when you did not solve it.

3. (20 pts) After sorting the attitude values to a monotonic order, your music teacher now wants to add a new presentation group with some attitude value a_{n+1}. Design an algorithm with $O(\log n)$ query complexity to insert the new presentation group to the right place such that the presentation groups are still sorted monotonically. Illustrate your algorithm with pseudo code, or even more preferably, with understandable words, and briefly explain why it runs within the demanded query complexity.

4. (Bonus 20 pts) Prove that it is impossible to find an algorithm with $o(n \log n)$ query complexity to sort the groups by their attitude values monotonically. Therefore, the result in sub-problem 3 is asymptotically optimal! *(Hint: Please check the textbook for the formal definition of the little-oh notation, if needed. You will get all bonus credits only if your proof is fully rigorous, while the TAs can choose to give partial credits.)*

A teacher once said

Students nowadays are extremely terrible.

In addition to the attitude values, now each group has a "terrible value" t_i. Similar to the attitude values, you do not know the exact values of $t_1, t_2, \cdots, t_n$, but you can call a similar function:

- `query_terrible_value(i,j,k)`: The parameters are three **distinct** indices $1 \leq i, j, k \leq n$. If $t_i \leq t_j \leq t_k$ or $t_i \geq t_j \geq t_k$, then this function will return TRUE; otherwise, it will return FALSE.

As every presentation group is unique, any two presentation groups i and j will have different attitude values ($a_i \neq a_j$) or different terrible values ($t_i \neq t_j$). Given the attitude values and terrible values, a good triplet is defined as a vector of three distinct indices (i, j, k) that satisfies both conditions below.

- ($a_i \leq a_j \leq a_k$ or $a_i \geq a_j \geq a_k$)
- ($t_i \leq t_j \leq t_k$ or $t_i \geq t_j \geq t_k$)

5. (10 pts) Suppose $n = 5$ and

$$\begin{aligned}(a_1, a_2, a_3, a_4, a_5) &= (3, 2, 1, 3, 4)\\ (t_1, t_2, t_3, t_4, t_5) &= (5, 1, 2, 3, 5)\end{aligned}$$

Please calculate the number of all the "good triplets." No more explanations are needed—we only need the result.

6. (30 pts) Design an algorithm that calculates the number of all the "good triplets." Note that a_i and t_i are not necessarily distinct, as illustrated with sub-problem 5. Illustrate your algorithm with pseudo code, or even more preferably, with understandable words. Prove the correctness of your algorithm and analyze its query complexity, where the query complexity is on the total call to the *two* black-box functions. Better query complexity results in a higher score.

Problem 3 - Argogo's Arrogance (100 pts)

Roosevelt Street is one of the busiest roads in the imaginary DSA City. The street was carefully planned and constructed such that every building on the street is of the same width. There are only buildings on the south side of the street, and the buildings are numbered consecutively with $0, 1, 2, \ldots$.

One day, Argogo, a diligent student in the DSA City, was walking down the north side of Roosevelt Street and admiring the magnificent street view across the road. After a while, she started to wonder: what will the street look like if all buildings are sorted according to their heights?

To answer this question, she went on the website of the DSA City government, and found out that the website provides data for the height of every building on Roosevelt Street. She transformed the height data to an integer array named `heights`, with the indices being their corresponding house numbers. For instance, `heights[0]` equals 5 means that building No. 0 is of height 5. She also stored the length of the array in a variable named `len`.

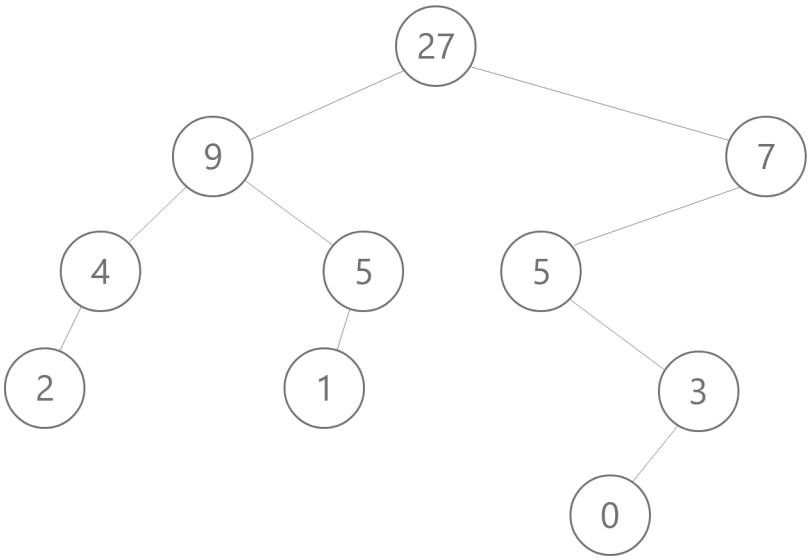
After acquiring the data , Argogo decided to incorporate the knowledge she learned from the DSA class to build a max heap efficiently for the data.

1. (15 pts) Consider `heights` to be $[2, 4, 9, 1, 5, 27, 5, 0, 3, 7]$, and build the max heap by inserting each element into the heap one by one. Draw the resulting heap. No more explanations are needed—we only need the resulting heap.

2. (15 pts) Consider the same `heights` array, but build the max heap by the Build-Max-Heap algorithm listed on page 157 of the textbook (if you do not have a textbook, we will provide the pseudo code on NTU COOL later). Draw the resulting heap. No more explanations are needed—we only need the resulting heap.

After seeing the heaps above, Argogo was able to spend the rest of the day happily with satisfaction. The next day, however, she thought about another issue: what is the tallest building in a particular section of the street? After doing some research, she discovered a beautiful data structure that can handle such an issue efficiently: Cartesian trees. The Cartesian tree is a type of binary tree that satisfies the properties of a max-tree (or min-tree), while preserving the order of the input sequence when conducting in-order traversal. That is, a Cartesian max-tree is a max-tree in its heights, and a binary search tree in its indices.

For example, the value sequence $[2, 4, 9, 1, 5, 27, 5, 0, 3, 7]$ can be transformed into the following Cartesian max-tree:



In particular, the tree satisfies the definition of a max-tree, and its in-order traversal outputs the original sequence.

3. (15pts) Design an algorithm that builds a Cartesian tree from an array `heights` with length `len` using $O(n)$ time, where n is `len`. Explain the algorithm (with pseudo code, or even more preferably, with understandable words) and why it meets the required time complexity.

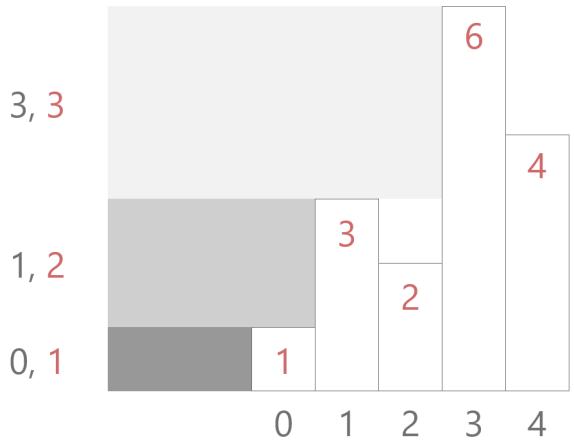
4. (15pts) Given `index`, design an algorithm that gets `height[index]` from the Cartesian tree built above using $O(\log n)$ time, assuming that the height of the tree is $O(\log n)$. Explain the algorithm (with pseudo code, or even more preferably, with understandable words) and why it meets the required time complexity. You can generally decorate the tree with $O(n)$ of additional space if needed.

5. (20pts) Given any `left` and `right`, design an algorithm that gets the largest value within `height[left], height[left+1], ..., height[right-1]` from the Cartesian tree built above using $O(\log n)$ time, assuming that the height of the tree is $O(\log n)$. Explain the algorithm (with pseudo code, or even more preferably, with understandable words) and why it meets the required time complexity. You can generally decorate the tree with $O(n)$ of additional space if needed.

Combining the `heights` data and the algorithms implemented above, Argogo spent the rest of the day having fun with the queries. On the next day, however, she decided to play with the

data more. She is curious about what the views would be like if we are at the left-hand-side (smaller-number side) of the street.

For example, for the `heights` sequence $[1, 3, 2, 6, 4]$, when viewing from the left-hand-side, we will only be able to see buildings $0, 1, 3$, with their visible heights $1, 2, 3$, respectively.



6. (20pts) Design an algorithm that outputs the left-hand-side view (the visible buildings and their heights) from the Cartesian tree built above using $O(\log n)$ time, assuming that the height of the tree is $O(\log n)$. Explain the algorithm (with pseudo code, or even more preferably, with understandable words) and why it meets the required time complexity.

Problem 4 - Teacher Lightning Bear's Kingdom (100 pts)

Problem Description

The Lightning Bear Kingdom is the greatest country in the world. There are N cities in the kingdom. Each city is represented by an integer between 1 and N, with city S being the capital of the kingdom. The cities are connected by roads, where each road is of unit length and connects two cities. Teacher Lightning Bear, the king of the Lightning Bear Kingdom, is the smartest person in the world. He designs the kingdom to be as simple as possible. In particular, he uses only $N-1$ roads to make all cities fully connected with each other. As a consequence, there is one and only one path for any pair of cities.

Now, Teacher Lightning Bear wants to enjoy his vacation at his favorite resort which is in city R. Thunder Tigger, a dear friend of Teacher Lightning Bear, wants to find where Teacher Lightning Bear is, in order to surprise him. Thunder Tigger thus asks you, the secretary of Teacher Lightning Bear, to provide some hints. Nevertheless, Teacher Lightning Bear does not really like surprises. If he finds that you leak the resort information to Thunder Tigger, he can get angry and you will certainly be fired. As a consequence, you can only communicate with Thunder Tigger with the following protocol.

- Thunder Tigger proposes city c as the query.
- You first write down the path from c to the capital city S as

$$(a_0 = c) \to a_1 \to a_2 \ldots \to (a_n = S)$$

- You then write down the path from c to the resort city R as

$$(b_0 = c) \to b_1 \to b_2 \ldots \to (b_m = R)$$

- Now you find the largest i such that $a_j = b_j$ for all $j \leq i$, and return a_i (or b_i, as $b_i = a_i$) to Thunder Tigger.

Note that it is totally possible that Teacher Lightning Bear is enjoying his vacation in the capital city. That is, $R = S$.

Thunder Tigger, who is very eager to know where Teacher Lightning Bear is, starts bouncing across the kingdom and querying you with many different c. You thus decide to write a program to help answer the queries faster.

Input

The first line of input contains four numbers N, Q, S, R, separated by space. They indicate the number of cities, the number of queries, the capital city, and the resort city, respectively.

Each of the next $N-1$ lines contains two integers a, b, separated by space, which indicates a (bi-directional) road between city a and city b.

Each of the next Q lines contains one integer c, indicating the query from Thunder Tigger.

Output

For each query, print a line that contains a single integer a_i, which means the answer that should be returned to Thunder Tigger.

Constraints

- $1 \leq N, Q \leq 10^6$
- $1 \leq a, b, c, S, R \leq N$
- All cities are connected with each other. (Equivalently, there is exactly one path for any pair of cities.)

Subtasks

Subtask 1 (15 pts)

- $1 \leq N, Q \leq 10$

Subtask 2 (20 pts)

- $1 \leq N, Q \leq 1000$

Subtask 3 (25 pts)

- $1 \leq N, Q \leq 10000$

Subtask 4 (40 pts)

- No other constraints

Sample Cases

Sample Input 1

```
4 4 1 4
1 2
2 3
3 4
1
2
3
4
```

Sample Output 1

```
1
2
3
4
```

Sample Input 2

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

Sample Output 2

```
4
2
1
1
1
```

Sample Input 3

```
10 3 7 7
6 1
2 8
1 10
7 9
5 1
9 8
8 6
4 6
3 10
7
8
1
```

Sample Output 3

```
7
7
7
```

Hints

The cities and roads can be viewed as a rooted tree from any city, such as c.

Problem 5 - Everybody Loves Brian (100 pts)

Problem Description

Brian is an intelligent and considerate student with many friends. His friends love asking him questions about stock analysis. In particular, each question is about the k-th "sweet point" among the inquiring friend's favorite stocks, which is defined as the k-th smallest price among those stocks within the last 10^9 days. Here the definition of the k-th smallest value is 1-based and takes repetition into account. That is, the 3rd smallest value within $\{1, 1, 2, 6\}$ is 2, and the 1st (& 2nd) smallest value is 1.

Answering each question efficiently, even with the help of computers, is a challenging task. Nevertheless, Brian has lots of background knowledge about stocks that can facilitate computation. One property for each stock, because of the inflation nature, is that its price will eventually rise after N days. Formally speaking, let $p(s, t)$ be the price for stock s on day t, then $p(s, t + n) > p(s, t)$ for all $n \geq N$.

In addition, Brian's friends often share similar interests to him in terms of the favorite stocks. Formally speaking, assume that the set of Brian's favorite stocks is $\mathbb{A} = \{s_1, s_2, \ldots, s_A\}$. Then the set of favorite stocks for each of Brian's friend would be either $\mathbb{A}$ or $\mathbb{A} \cup \{s\}$, where s is an extra stock that is not in $\mathbb{A}$.

Given the properties above, can you write a program to help Brian efficiently answer the many questions from his friends?

Input

The first line contains three integers A, Q, N, separated by space.

- A denotes the number of Brian's favorite stocks
- Q denotes the number of questions
- N defines how long each stock is guaranteed to increase its price, as mentioned above

The second line contains A integers $s_1, s_2, ...s_A$, separated by space.

- s_i denotes the i-th stock in $\mathbb{A}$

Each of the following Q lines contains two integers s and k, separated by space.

- s denote the extra stock chosen by the inquiring friend, where $s = 0$ means that no extra stock is chosen.

Output

For each question, you should print one line, containing an integer that represents the k-th "sweet point" for that question.

Constraints

- $0 \leq s, s_i \leq 10^9$ and $s_i \neq 0$
- $1 \leq k \leq 10^6$
- $1 \leq Q,\ A,\ N \leq 1024$

Important Notes

You are given `price.h` on NTU COOL, which contains the function `price(s, t)` that helps you calculate the price of stock s at day t. You can simply download and `#include "price.h"` within your code, and there is no need to understand or modify the code in order to get accepted. We guarantee that:

- You can expect `price(s, t)` to run in constant time. Additionally, you can assume that you can run at least 10^7 times of `price(s, t)` per second on the judge.
- The valid input range of `price(s, t)` is $1 \leq s, t \leq 10^9$, and the output range is within `unsigned long long`. Inputs that are out of range cause undefined behaviors.
- The private `price.h` on the judge system may not be exactly the same as the public one on your hand. So you should only use the properties illustrated above to solve your problem, rather than relying on any other hacks within the public `price.h`. Nevertheless, you are welcomed to check the public `price.h` to understand how such a (random) function can be constructed.

Subtasks

Subtask 1 (15 pts)

- $s = 0$
- $A = 1$
- $N = 1$

Subtask 2 (15 pts)

- $s = 0$
- $N = 1$

Subtask 3 (15 pts)

- $s = 0$
- $A = 1$

Subtask 4 (15 pts)

- $s = 0$

Subtask 5 (15 pts)

- $N = 1$

Subtask 6 (25 pts)

No other constraints

Sample Cases

Sample Input 1

```
1 2 1
1
0 1
0 1000000
```

Sample Output 1

```
8
2448802476
```

Sample Input 2

```
9 3 1
1 2 3 4 5 6 7 8 9
0 1000000
0 16384
0 121
```

Sample Output 2

```
19841729
29667
59
```

Sample Input 3

```
1 2 1000
1000000000
0 1000000
0 333333
```

Sample Output 3

```
18434768613
12617963493
```

Sample Input 4

```
3 3 1000
999999991 999999992 999999993
0 1000000
999999994 1000000
900000005 123456
```

Sample Output 4

```
24875743063
11851171780
144499628
```

Hints

Here are some friendly hints. You are of course welcomed to challenge yourself on solving the problem without checking those hints.

1. You can first test with simple functions instead of the given `price(s, t)` to simulate the solution with your brain more easily. For instance, $p(s,t) = s + t$ or even $p(s,t) = t$ can be good choices.

2. The subtasks address some different concepts but are not necessarily ordered by difficulty. It is recommended to solve each subtask separately first.

3. For each subtask, you can consider solving with $Q = 1$ before moving on to the multi-question case.

4. Subtasks 1, 2 and 5 come with $N = 1$. It can be helpful to thinking about what the price function would look like when $N = 1$ before moving to bigger N.