B — Better and faster!

You probably know this story already. You wake up in the morning and your head feels twice the size. You have a vague memory of a program your boss asked you to write. After you have logged in, you see a main piece of code you wrote yesterday.

```
unsigned int checksum (char str[], int len) {
    unsigned int r = 0;
    for (int k=0; k<8*len; k++) {               // iterate over bits of str
        if ((r & (1<<31)) != 0) r = (r << 1) ^ 0x04c11db7;
                           else r = (r << 1);   // do some magic
        if (str[k/8] & 1<<(7-k%8))              // if the k-th bit of str is set,
            r ^= 1;                             //   then flip the last bit of r
    }
    return r;
}
```

"Good", you think, "I commented it well". Still, you have some issues with understanding the "do some magic" part. But well, the function is called `checksum`, and — lo and behold — it really computes a kind of a checksum of a given string.

You recall the rest of your task. You were supposed to compute this checksum for a given string and then for slightly modified versions of this string. Actually, the rest of your program also looks quite decent.

```
#include <stdio.h>

int main()
{
    char str[1000001],c;
    int TESTS,n,changes,p;
    for (scanf ("%d", &TESTS); TESTS>0; TESTS--) {
        scanf ("%d %s", &n, str);               // read the input
        printf ("%u\n", checksum(str, n));      // compute checksum for original string
        for (scanf ("%d", &changes); changes>0; changes--) {
            scanf ("%d %c", &p, &c);            // apply the change
            str[p-1] = c;
            printf ("%u\n", checksum(str, n));  // compute checksum for modified string
        }
    }
}
```

And then you recall the final issue. The program works perfectly well, but also terribly slow. You just have to make it work faster. Much faster. As you have heard that Java is a better and safer programming language, you even made an equivalent Java version (see the last page), which works even slower (strange, eh?).

Multiple Test Cases

The input contains several test cases. The first line of the input contains a positive integer $Z \leq 20$, denoting the number of test cases. Then Z test cases follow, each conforming to the format described in section *Single Instance Input*. For each test case, your program has to write an output conforming to the format described in section *Single Instance Output*.

Single Instance Input

Below by a *character*, we mean a single small or large letter, or a digit.

In the first line of an input instance, there is a natural number n ($1 \leq n \leq 10^6$) and a string s, separated by a single space. String s consists of n characters. The second line of the input contains

one integer t ($0 \leq t \leq 10^5$) denoting the number of changes to be applied to string s. Each of the next t lines consists of a natural number $p \in [1, n]$ and a character c, separated by a single space. It encodes a change of a string: the p-th character of s has to be replaced by c.

Single Instance Output

You have to produce the same output the program above would do. In other words, you have to output $t+1$ lines, each containing a natural number being a checksum. The first checksum has to be computed for an original string s, the remaining ones are to be computed after each change made to s.

Example

Input	Output
1 5 ABcd3 3 1 B 2 A 1 d	1914964467 2137468714 2087137066 4274181240

Java Version of The Program

```
import java.util.Scanner;

public class Compute {

static long checksum (byte[] str, int len) {
    int r = 0;
    for (int k=0; k<8*len; k++) {                  // iterate over bits of str
        if ((r & (1<<31)) != 0) r = (r << 1) ^ 0x04c11db7;
                           else r = (r << 1);      // do some magic
        if ((str[k/8] & 1<<(7-k%8)) != 0)          // if the k-th bit of str is set,
            r ^= 1;                                //   then flip the last bit of r
    }
    long rr = (r<0 ? r+0x100000000L : r);          // Java does not have unsigned int
    return rr;
}

public static void main(String[] args) {
    Scanner in = new Scanner(System.in);
    for (int TESTS = in.nextInt(); TESTS>0; TESTS--) {
        int n = in.nextInt();                      // read the input
        byte[] str = in.next().getBytes();         // compute checksum for
        System.out.println (checksum(str,n));      //   original string
        for (int changes = in.nextInt(); changes>0; changes--) {
            int p = in.nextInt();                  // apply the change
            byte c = in.next().getBytes()[0];
            str[p-1] = c;
            System.out.println (checksum(str,n));  // compute checksum for
        }                                          //   modified string
    }
}
```