Educational Codeforces Round 81 (Rated for Div. 2)

A. Display The Number

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

You have a large electronic screen which can display up to 998244353 decimal digits. The digits are displayed in the same way as on different electronic alarm clocks: each place for a digit consists of 7 segments which can be turned on and off to compose different digits. The following picture describes how you can display all 10 decimal digits:

As you can see, different digits may require different number of segments to be turned on. For example, if you want to display 1, you have to turn on 2 segments of the screen, and if you want to display 8, all 7 segments of some place to display a digit should be turned on.

You want to display a really large integer on the screen. Unfortunately, the screen is bugged: no more than n segments can be turned on simultaneously. So now you wonder what is the greatest integer that can be displayed by turning on no more than n segments.

Your program should be able to process t different test cases.

Input

The first line contains one integer t ($1 \le t \le 100$) — the number of test cases in the input.

Then the test cases follow, each of them is represented by a separate line containing one integer n ($2 \le n \le 10^5$) — the maximum number of segments that can be turned on in the corresponding testcase.

It is guaranteed that the sum of n over all test cases in the input does not exceed 10^5.

Output

For each test case, print the greatest integer that can be displayed by turning on no more than n segments of the screen. Note that the answer may not fit in the standard 32-bit or 64-bit integral data type.

Example

input
2 3 4
output
7 11

B. Infinite Prefixes

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

You are given string s of length n consisting of 0-s and 1-s. You build an infinite string t as a concatenation of an infinite number of strings s, or $t = ssss\ldots$. For example, if $s =$ 10010, then $t =$ 100101001010010...

Calculate the number of prefixes of t with *balance* equal to x. The balance of some string q is equal to $cnt_{0,q} - cnt_{1,q}$, where $cnt_{0,q}$ is the number of occurrences of 0 in q, and $cnt_{1,q}$ is the number of occurrences of 1 in q. The number of such prefixes can be infinite; if it is so, you must say that.

A prefix is a string consisting of several first letters of a given string, without any reorders. An empty prefix is also a valid prefix. For example, the string "abcd" has 5 prefixes: empty string, "a", "ab", "abc" and "abcd".

Input

The first line contains the single integer T ($1 \le T \le 100$) — the number of test cases.

Next $2T$ lines contain descriptions of test cases — two lines per test case. The first line contains two integers n and x ($1 \le n \le 10^5$, $-10^9 \le x \le 10^9$) — the length of string s and the desired balance, respectively.

The second line contains the binary string s ($|s| = n$, $s_i \in \{0, 1\}$).

It's guaranteed that the total sum of n doesn't exceed 10^5.

Output
Print T integers — one per test case. For each test case print the number of prefixes or -1 if there is an infinite number of such prefixes.

Example

input
4 6 10 010010 5 3 10101 1 0 0 2 0 01
output
3 0 1 -1

Note
In the first test case, there are 3 good prefixes of t: with length 28, 30 and 32.

C. Obtain The String

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

You are given two strings s and t consisting of lowercase Latin letters. Also you have a string z which is initially empty. You want string z to be equal to string t. You can perform the following operation to achieve this: append any subsequence of s at the end of string z. A subsequence is a sequence that can be derived from the given sequence by deleting zero or more elements without changing the order of the remaining elements. For example, if $z = ac$, $s = abcde$, you may turn z into following strings in one operation:

1. $z = acace$ (if we choose subsequence ace);
2. $z = acbcd$ (if we choose subsequence bcd);
3. $z = acbce$ (if we choose subsequence bce).

Note that after this operation string s doesn't change.

Calculate the minimum number of such operations to turn string z into string t.

Input
The first line contains the integer T ($1 \le T \le 100$) — the number of test cases.

The first line of each testcase contains one string s ($1 \le |s| \le 10^5$) consisting of lowercase Latin letters.

The second line of each testcase contains one string t ($1 \le |t| \le 10^5$) consisting of lowercase Latin letters.

It is guaranteed that the total length of all strings s and t in the input does not exceed $2 \cdot 10^5$.

Output
For each testcase, print one integer — the minimum number of operations to turn string z into string t. If it's impossible print -1.

Example

input
3 aabce ace abacaba aax ty yyt
output
1

```
-1
3
```

D. Same GCDs

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

You are given two integers a and m. Calculate the number of integers x such that $0 \le x < m$ and $\gcd(a, m) = \gcd(a + x, m)$.

Note: $\gcd(a, b)$ is the greatest common divisor of a and b.

Input
The first line contains the single integer T ($1 \le T \le 50$) — the number of test cases.

Next T lines contain test cases — one per line. Each line contains two integers a and m ($1 \le a < m \le 10^{10}$).

Output
Print T integers — one per test case. For each test case print the number of appropriate x-s.

Example

input
3 4 9 5 10 42 9999999967
output
6 1 9999999966

Note
In the first test case appropriate x-s are $[0, 1, 3, 4, 6, 7]$.

In the second test case the only appropriate x is 0.

E. Permutation Separation

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

You are given a permutation $p_1, p_2, \ldots, p_n$ (an array where each integer from 1 to n appears exactly once). The weight of the i-th element of this permutation is a_i.

At first, you separate your permutation into two **non-empty** sets — prefix and suffix. More formally, the first set contains elements $p_1, p_2, \ldots, p_k$, the second — $p_{k+1}, p_{k+2}, \ldots, p_n$, where $1 \le k < n$.

After that, you may move elements between sets. The operation you are allowed to do is to choose some element of the first set and move it to the second set, or vice versa (move from the second set to the first). You have to pay a_i dollars to move the element p_i.

Your goal is to make it so that each element of the first set is less than each element of the second set. Note that if one of the sets is empty, this condition is met.

For example, if $p = [3, 1, 2]$ and $a = [7, 1, 4]$, then the optimal strategy is: separate p into two parts $[3, 1]$ and $[2]$ and then move the 2-element into first set (it costs 4). And if $p = [3, 5, 1, 6, 2, 4]$, $a = [9, 1, 9, 9, 1, 9]$, then the optimal strategy is: separate p into two parts $[3, 5, 1]$ and $[6, 2, 4]$, and then move the 2-element into first set (it costs 1), and 5-element into second set (it also costs 1).

Calculate the minimum number of dollars you have to spend.

Input
The first line contains one integer n ($2 \le n \le 2 \cdot 10^5$) — the length of permutation.

The second line contains n integers $p_1, p_2, \ldots, p_n$ ($1 \le p_i \le n$). It's guaranteed that this sequence contains each element from 1 to n exactly once.

The third line contains n integers $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le 10^9$).

Output
Print one integer — the minimum number of dollars you have to spend.

Examples

input
3 3 1 2 7 1 4
output
4

input
4 2 4 1 3 5 9 8 3
output
3

input
6 3 5 1 6 2 4 9 1 9 9 1 9
output
2

F. Good Contest

time limit per test: 3 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

An online contest will soon be held on ForceCoders, a large competitive programming platform. The authors have prepared n problems; and since the platform is very popular, 998244351 coder from all over the world is going to solve them.

For each problem, the authors estimated the number of people who would solve it: for the i-th problem, the number of accepted solutions will be between l_i and r_i, inclusive.

The creator of ForceCoders uses different criteria to determine if the contest is good or bad. One of these criteria is the number of inversions in the problem order. An inversion is a pair of problems (x, y) such that x is located earlier in the contest ($x < y$), but the number of accepted solutions for y is **strictly** greater.

Obviously, both the creator of ForceCoders and the authors of the contest want the contest to be good. Now they want to calculate the probability that there will be **no** inversions in the problem order, assuming that for each problem i, any **integral** number of accepted solutions for it (between l_i and r_i) is equally probable, and all these numbers are independent.

Input

The first line contains one integer n ($2 \le n \le 50$) — the number of problems in the contest.

Then n lines follow, the i-th line contains two integers l_i and r_i ($0 \le l_i \le r_i \le 998244351$) — the minimum and maximum number of accepted solutions for the i-th problem, respectively.

Output

The probability that there will be no inversions in the contest can be expressed as an irreducible fraction $\frac{x}{y}$, where y is coprime with 998244353. Print one integer — the value of xy^{-1}, taken modulo 998244353, where y^{-1} is an integer such that $yy^{-1} \equiv 1 \pmod{998244353}$.

Examples

input
3 1 2 1 2 1 2
output
499122177

input
2 42 1337 13 420
output
578894053

input
2 1 1 0 0
output
1

input
2 1 1 1 1
output
1

Note

The real answer in the first test is $\frac{1}{2}$.