Problem A. Astronomy Problem

Input file:	`astronomy.in`
Output file:	`astronomy.out`
Time limit:	2 seconds
Memory limit:	512 megabytes

Archeologists of Flatland planet have found several ancient manuscripts. Each manuscript describes the location of three parts of some powerful artifact. The parts of each artifact are hidden at Flatland and two other planets. The planets are not identified in the manuscript, but it is said that the distance from Flatland to one of the planets is $\sqrt{a_i}$, the distance from Flatland to another planet is $\sqrt{b_i}$ and the distance between the planets is $\sqrt{c_i}$.

Flatland universe is a plane with orthogonal Cartesian coordinate system. Flatland is located at a point $(0, 0)$. There are n planets except Flatland in the universe, the i-th planet is located at a point (x_i, y_i).

Help archeologists to solve astronomy problem, for each manuscript find the number of possible pairs of planets that satisfy the manuscript description.

Input

The input file contains multiple test cases.

The first line of each test case contains n — the number of other planets ($1 \le n \le 3000$). The following n lines contain two integers each: x_i, y_i ($-10^9 \le x_i, y_i \le 10^9$). No two planets coincide. No other planet is located at $(0, 0)$.

The following line contains m — the number of manuscripts to process ($1 \le m \le 3000$). The following m lines contain three integers each: a_i, b_i, c_i ($1 \le a_i, b_i, c_i \le 8 \cdot 10^{18}$).

Input is followed by a line with $n = 0$.

The sum of values of n in all test cases in one file doesn't exceed 3000. The sum of values of m in all test cases in one file doesn't exceed 3000.

Output

For each test case output m integers: for each manuscript output the number of planet pairs that satisfy the description from the manuscript.

Examples

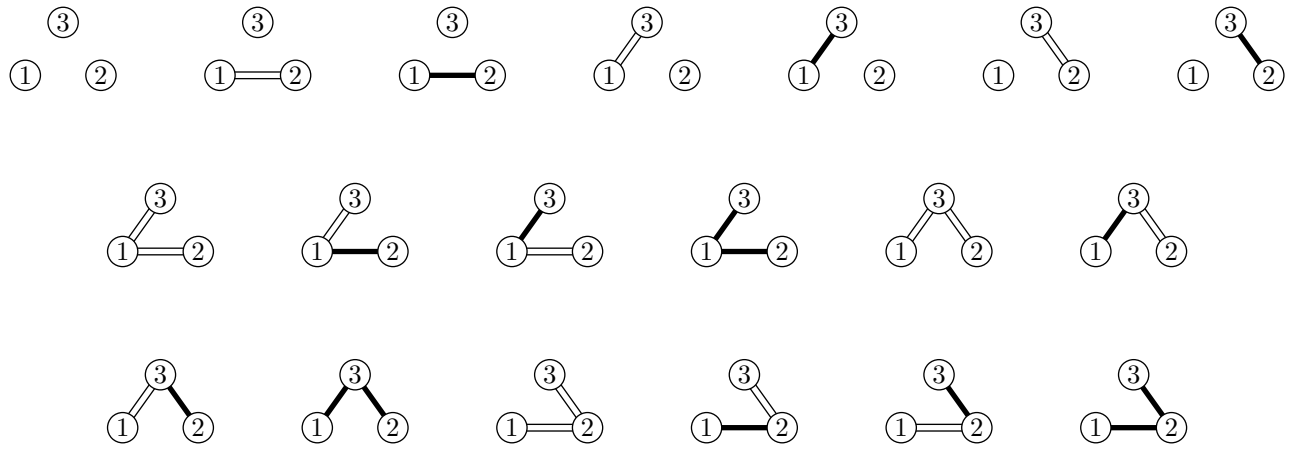
astronomy.in	astronomy.out
6 0 2 1 1 2 0 5 0 -3 4 -4 3 4 25 25 2 4 2 2 4 25 9 25 25 100 0	1 2 1 0

Problem B. Bipartite Bicolored Graphs

Input file:	`bipartite.in`
Output file:	`bipartite.out`
Time limit:	2 seconds
Memory limit:	512 megabytes

The undirected graph G is called *bipartite* if the set of its vertices V can be partitioned into two disjoint subsets $V = X \cup Y$ such that for each edge uv in G its ends u and v belong to different subsets. Graph is called *bicolored* if all of its edges are colored into two colors: black and white.

Given n, find the number of labeled bipartite bicolored graphs with n vertices. The picture below shows all 19 such graphs for $n = 3$. The answer can be quite large, so you have to find it modulo 175 781 251.



Input

The input file contains multiple test cases.

Each test case contains a single integer n on a line by itself ($1 \le n \le 100$).

Input is followed by a line with $n = 0$.

Output

For each test case output one integer: the number of labeled bipartite bicolored graphs with n vertices.

Examples

bipartite.in	bipartite.out
1 2 3 0	1 3 19

Problem C. Catalan Numbers

Input file:	`catalan.in`
Output file:	`catalan.out`
Time limit:	2 seconds
Memory limit:	512 megabytes

Andrew likes Catalan numbers. Also Andrew likes to joke.

He is an experienced problem setter and prepares lots of contests for training camps. Each contest he prepares a problem that has one integer as input, one integer as output, and answers for 0, 1, 2, 3, 4 and 5 are, respectively, 1, 1, 2, 5, 14 and 42. However, answers for greater inputs don't coincide with corresponding Catalan numbers.

Andrew has already prepared so many contests, that he is short of good problems with such property. So he decided to automate the process of creating such problems. As a good pool of possible problems he considers problems of counting words of specific length in deterministic finite automata. Andrew has chosen k — the desired answer for the input 6, and wants to find deterministic finite automaton that accepts 1, 1, 2, 5, 14, 42, k words of length 0, 1, 2, 3, 4, 5, 6, respectively, that has minimal number of states.

Recall, that the deterministic finite automaton (DFA) is an ordered set $\langle \Sigma, U, S, T, \varphi \rangle$ where Σ is the finite set called *input alphabet*, U is the finite set of *states*, $S \in U$ is the *initial state*, $T \subset U$ is the set of *terminal states* and $\varphi : U \times \Sigma \to U \cup \{\varnothing\}$ is the *transition function*.

The input of the automaton is the string α over Σ. Initially the automaton is in state s. Each step the automaton reads the first character c of the input string and changes its state to $\varphi(u, c)$ where u is the current state. If $\varphi(u, c) = \varnothing$ the automaton immediately rejects α. Then the first character of the input string is removed and the step repeats. If after its input string is empty the automaton is in the terminal state, it accepts the initial string α, in the other case it rejects it.

You are given an integer k. Construct deterministic finite automaton with alphabet of size at most 20, such that it has minimal number of states, and the number of words of lengths from 1 to 6 accepted by the automaton is given by the following table. The number of longer accepted words can be arbitrary.

length	number of accepted words
0	1
1	1
2	2
3	5
4	14
5	42
6	k

Input

The input file contains multiple test cases.

Each test case contains a single integer k on a line by itself ($120 \le k \le 140$).

Input is followed by a line with $n = 0$.

Output

For each test case print the requested deterministic automaton in the following format. The first line must contain two integers: n — the number of states and s — the size of the alphabet ($1 \le s \le 20$). Note that you must minimize n, but you don't have to minimize s.

Let the states be numbered from 1 to n, the starting state be 1, let the characters of the alphabet be numbered from 1 to s.

The second line must contain n integers and specify whether corresponding states are terminal: i-th of them must be 1 if the i-th state is terminal, or 0 if the i-th state is not terminal.

The following n lines must contain s integers each: the j-th of the i-th line must be the value of $\varphi(i, j)$ — the state where the transition from the i-th state by the j-th character goes, or 0 if $\varphi(i, j) = \varnothing$.

Examples

catalan.in	catalan.out
131 0	3 4 1 0 0 1 2 0 0 1 2 2 3 2 3 3 0

In the given example if $\Sigma = \{a, b, c, d\}$ the five words of length 3 accepted by the automaton are "aaa", "aba", "baa", "bba" and "bca".

Problem D. Dichromatic Trees

Input file:	dichromatic.in
Output file:	dichromatic.out
Time limit:	20 seconds
Memory limit:	512 megabytes

Dichromatic trees, currently better known as red-black trees, is a data structure for ordered set originally invented by Rudolf Bayer in 1972 and later improved by Leonidas Guibas and Robert Sedgewick.

Red-black tree is a binary rooted tree where each vertex is colored either red or black. Each vertex can have a left child and a right child. The following conditions are satisfied:

- Children of the red vertex are black.
- Consider path from the root to any position where the child of some vertex is missing. Any such path must contain the same number of black vertices. This number is called the *black height* of the tree.

Given n and h find the number of red-black trees with n vertices that have black height at most h. Note that trees that have the same structure but differ by vertex coloring are considered different. Output the answer modulo 258 280 327.

Input

The input file contains k test cases. All test cases in one input file share the same h.

The first line of the input file contains two integers: k and h ($1 \le k \le 131\,072$, $0 \le h \le 16$). The second line contains k integers: $n_1, n_2, \ldots, n_k$ ($1 \le n_i \le 131\,072$).

Output

Output k numbers, for each i from 1 to k output the number of red-black trees with n_i vertices that have black height at most h.

Examples

dichromatic.in	dichromatic.out
10 2 1 2 3 4 5 6 7 8 9 10	2 2 3 8 14 20 34 56 90 164

Problem E. Ebola Virus

Input file:	ebola.in
Output file:	ebola.out
Time limit:	2 seconds
Memory limit:	512 megabytes

You are the leader of the team that is going to cure Ebola virus using new experimental medicine in several villages of one African country. There are n villages located along the only road of the country, numbered from 1 to n along the road. In the morning of the first day you are at the village number 1.

Initially there are a_i people who are infected with Ebola virus in the i-th village. Each day you can do one of the following: either cure all infected people in the village you are currently in, or move to neighboring village. If there are still k infected people in some village by the end of the day, they infect k other people in the village and then die. So actually the number of infected people stays the same in each uncured village, but each day a_i new people die.

You would like to cure people in all villages in such way that as few people as possible eventually die. However, there is a restriction: people must not lose hope. So sometimes your actions are forced by the following rule. Suppose that you have entered the i-th village and decided not to cure people in it, but go to the other village on the following day. In such case if afterwards you decide to move from village j to village j' such that j' is closer to i than j, you must keep moving in this direction until you reach village i, and you must cure people in i when reaching it. You may, however, stop to cure people in intermediate villages while doing so.

For example, suppose that you are initially in the village 1. On the first day you move to village 2. On the second day you move to village 3. On the third day you cure people in village 3. On the fourth day you move to village 2. Now you are forced to cure people in village 2 by the rule above. Also you are not allowed to move to village with greater number until you reach and cure village 1. So you must cure people in village 2 on day 5, move to village 1 on day 6 and cure people there on day 7. Now you again can choose your actions.

Input

The input file contains multiple test cases.

The first line of each test case contains n — the number of villages ($1 \le n \le 3000$). The following line contains n integers: $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le 10^9$). Input is followed by a line with $n = 0$.

The sum of values of n in all test cases in one file doesn't exceed 3000.

Output

For each test case output one integer: the minimal possible number of people that would die before people in all villages are cured.

Examples

ebola.in	ebola.out
6 40 200 1 300 2 10 0	1950

In the given example the optimal sequence of actions is the following: go to village 2, cure people at village 2, go to village 3, go to village 4, cure people at village 4, go to village 3, cure people at village 3 (forced), go to village 2 (forced), go to village 1 (forced), cure people at village 1 (forced), go to villages 2, 3, 4, 5, 6, cure people at village 6, go to village 5, cure people at village 5 (forced, though need to do it any way).

The total curing process takes 18 days, number of people dying each day is, respectively, 553, 353, 353, 353, 53, 53, 52, 52, 52, 12, 12, 12, 12, 12, 12, 2, 2, 0.

Problem F. Figure Skating

Input file:	`figure.in`
Output file:	`figure.out`
Time limit:	1 second
Memory limit:	512 megabytes

Martian Federation of Figure Skating has adapted the new rules for Mars Figure Skating Championship that will take part in Matrozavodsk on August 31.

The *program* of each competitor consists of a series of *jumps*. There are four jumps that Martians can perform: Axel, Flip, Lutz and Salchow. The program is considered *beautiful* by the judges if it satisfies the following conditions. In all descriptions *arbitrary number of jumps* means 0 or more jumps.

- A *beautiful program* is a *good sequence* of jumps.
- A *good sequence* of jumps starts with an arbitrary number of Axel jumps. After that the sequence can be over, or it can continue with a Flip followed by either *romantic sequence* or *energetic sequence*.
- A *romantic sequence* of jumps starts with arbitrary number of Flip or Lutz jumps (not necessarily all the same) followed by
  - a Lutz jump or a Salhow jump and then an *energetic sequence* or
  - an Axel jump and then a *good sequence*.
- An *energetic sequence* of jumps starts with arbitrary number of Flip jumps followed by a Salhow jump and then a *tragic sequence*.
- A *tragic sequence* of jumps starts with arbitrary number of Flip jumps followed by
  - an Axel jump and then a *romantic sequence*.
  - an Axel jump or a Salhow jump and then an *energetic sequence*.

The rules are so complicated that the judges wonder how many ways are there to create different beautiful programs of exactly n jumps. Help them find that out. The number can be quite large, so output it modulo 998 244 353.

Input

The input file contains multiple test cases.

Each test case consists of a single line that contains n ($1 \le n \le 10^9$).

The last test case is followed by a line containing a single zero.

There are at most 1000 test cases.

Output

For each test case output one integer — the number of jump sequences that make a beautiful program. Output the answer modulo 998 244 353.

Examples

figure.in	figure.out
1 2 3 4 5 0	1 2 5 14 42

There are 5 sequences of 3 jumps that make a beautiful program: ⟨Axel, Axel, Axel⟩; ⟨Axel, Flip, Axel⟩; ⟨Flip, Axel, Axel⟩; ⟨Flip, Flip, Axel⟩; ⟨Flip, Lutz, Axel⟩.

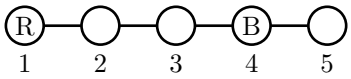
Problem G. Game of Col on Bamboo Forests

Input file:	game.in
Output file:	game.out
Time limit:	2 seconds
Memory limit:	512 megabytes

Game of Col is a map coloring game described by John Conway who attributed it to Colin Vout.

The Game is played by two players, Alice and Bob. The game is played on a graph, players make moves in turn by coloring graph vertices, Alice moves first. Each move a player chooses one of yet uncolored vertices and colors it to his/her color. Alice colors vertices red, Bob colors vertices blue. The restriction is that the player is not allowed to color vertices that are adjacent to vertices of his color. The player who cannot make a move loses.

For example, in the position on the graph below (red vertices are marked with "R", blue vertices are marked with "B") Alice can make her move by coloring vertex 3 or vertex 5. If she colors vertex 3, Bob can then color vertex 2. Then Alice colors vertex 5, Bob has no moves and loses. However, should the players played optimally on the same graph, Bob can win regardless of Alice's moves (check it!).



Alice and Bob have decided that they will play the game on bamboo forests. Bamboo forest is a graph that has k connected components, the i-th component is a path — bamboo. The picture below shows bamboo forest composed of bamboos of lengths 2, 2, and 4.

They have prepared n bamboos of length $a_1, a_2, \ldots, a_n$, respectively, and now they want to choose k of them to create a graph and play. Bob thinks that he would win any way, so he allows Alice to choose k bamboos to create a graph to play. Alice wonders, how many ways are there to choose k of the given bamboos, so that she would win the game if played optimally. Help her find the answer and print it modulo 242 121 643.

Input

The input file contains multiple test cases.

The first line of each test case contains n and k ($1 \le k \le n \le 100$). The second line contains n integers: $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le 10^9$).

The last test case is followed by a line containing two zeroes, it must not be processed. Each input file contains at most 1000 test cases.

Output

For each test case output one integer: the number of ways to choose k bamboos of the given n so that Alice wins the game of Col on the resulting bamboo forest. The answer must be printed modulo 242 121 643.

Examples

game.in	game.out
5 3 1 1 3 4 5 4 2 2 2 2 2 0 0	6 0

Problem H. Heavy-Light Decomposition

Input file:	heavylight.in
Output file:	heavylight.out
Time limit:	2 seconds
Memory limit:	512 megabytes

Heavy-Light decomposition is the way of partitioning a tree to a set of paths such that walking from any vertex to the root of the tree requires $O(\log n)$ changes between paths. In this problem you have to maintain heavy-light decomposition of a binary tree which has some of its leaves removed one after one.

Consider a binary tree T with n vertices numbered from 1 to n. Each vertex can have a left child and a right child. Vertex number 1 is the root of the tree, it is not a child of another vertex. Each other vertex is the child of some vertex. Vertex that has no children is called a *leaf*. For each vertex the size of this vertex is the number of vertices in its subtree. An edge that leads from vertex u to its child v is said to be *heavy* if the size of v is greater than the size of another u's child w or if v is the only child of u. If u has two children and both of them have the same size, initially the heavy edge from u leads to its left child.

First you must find all heavy edges in the tree and print the sum of numbers of vertices they lead to. After that you must process m requests: remove leaf vertex u_i from the tree, update all heavy edges, and print the sum of numbers of vertices that they lead to now. If after removing the vertex from the tree some vertex u has children of equal sizes, heavy edge from u is not changed.

Input

The input file contains several test cases.

The first line of each test case contains n — the number of vertices in a tree ($2 \le n \le 200\,000$). The following n lines contain description of the tree. The i-th of these lines contains two integers: L_i and R_i — numbers of left and right children of the i-th vertex, or 0 if the i-th vertex has no corresponding child.

The following line contains m — the number of leaves to be removed ($1 \le m \le n - 1$). The following line contains m integers: $u_1, u_2, \ldots, u_m$ — the vertices to remove. It is guaranteed that after all previous removals, the vertex u_i is a leaf.

Input is terminated by a line that contains $n = 0$, it must not be processed. The sum of values of n in all tests in one input file doesn't exceed 200 000.

Output

For each test case output $m + 1$ integers. The first of them must be the sum of vertices that heavy edges lead to in the initial tree. The following numbers must be equal to this sum after the corresponding leaves have been deleted.

Examples

heavylight.in	heavylight.out
8 2 3 4 5 0 0 6 7 0 8 0 0 0 0 0 0 7 6 7 8 5 4 2 3 0	20 21 15 7 6 2 3 0

Problem I. Interactive Memory Management

Input file:	`standard input`
Output file:	`standard output`
Time limit:	2 seconds
Memory limit:	512 megabytes

This is interactive problem.

In this problem you must implement the primitive memory manager that can allocate and free blocks of memory. Memory is represented as an array $M[1..n]$ of n cells. *Block* of size s consists of s consecutive cells of memory. Different blocks are not allowed to share the same cell.

Your memory manager must be able to allocate blocks of size 1, 2 and 3, free previously allocated blocks, and it can move previously allocated blocks.

There are two types of requests: allocate memory block and free memory block. All requests are numbered consequently starting from 1.

Before allocating a new block and after freeing a block you are allowed to make up to two 2 moves of currently allocated blocks.

Allocate request is specified as one integer from the set $\{1, 2, 3\}$ — size of the block to allocate.

Free request is specified as one negative integer. An integer $-k$ means that the block allocated at request k must be freed.

There are three types of operations that your memory manager can do.

- *allocate*: "`A` x" allocate requested block starting from cell number x.
- *move*: "`M` x y" move block starting from position x to start from position y.
- *done*: "`D`" report that moving blocks after performing free request is completed.

You have to implement memory manager that would perform all operations, given that at any moment the total size of allocated blocks is at most $n - 1$.

Interaction Protocol

First your program must read n — size of memory array ($4 \le n \le 100\,000$). After that one or more requests follow.

Each request is specified as an integer. Positive integer from 1 to 3 specifies *allocate* request, negative integer specifies *free* request. Zero specifies end of input, it must not be processed, after reading zero your program must exit.

After reading *allocate* request your program can perform up to 2 *move* operations, followed by *allocate* operation.

After reading *free* request your program must immediately free cells occupied by the corresponding block (it must not be reported), then it can perform up to 2 *move* operations, and then perform *done* operation.

All performed operations must be correct.

For each *allocate* when allocating block of size s its argument x must be index of the cell such that cells $x, \ldots, x + s - 1$ are free.

For each *move* its first argument x must be the first cell of some currently allocated block. If such block has size s, the second argument y must be index of the cell such that cells $y, \ldots, y + s - 1$ are free. In particular, source and destination locations of the block being moved may not share a common cell.

All requests are correct: no block is freed more than once, at any moment the total size of all allocated blocks is at most $n - 1$, each *free* request refers to the number of some *allocate* request.

It is guaranteed that there is strategy that correctly fulfils all requests.

There are at most 30 000 requests.

Examples

standard input	standard output
7 3 2 -1 1 3 0	 A 1 A 4 D A 1 M 1 6 A 1

Problem J. Jingles of a String

Input file:	jingles.in
Output file:	jingles.out
Time limit:	2 seconds
Memory limit:	512 megabytes

Given a string s a *jingle* $J(L, R)$ of its substring $s[L..R]$ is the set of characters that appear among its characters. For example, if $s =$ "abacaba", then $s[3..5] =$ "aca" and $J(3, 5) = \{a, c\}$.

You are given a string s. Find all possible jingles of its non-empty substrings and for each possible jingle find the longest substring of s with such jingle.

In the example above, there are 6 sets that appear as jingles of substrings of s. For example, for a set $\{a, b\}$ there are six substrings that has such jingle: $s[1..2]$, $s[1..3]$, $s[2..3]$, $s[5..6]$, $s[5..7]$, $s[6..7]$. Among them $s[1..3]$ (as well as $s[5..7]$) has length 3, this is the longest substring of s with such jingle.

Since printing all jingles would make output file too large, you have to output the sum

$$v(s) = \sum_{J \in \mathbb{J}} S(J)L(J).$$

Here $\mathbb{J}$ is the set of all jingles of the string, $S(J)$ is the number of characters in J, $L(J)$ is the length of longest substring of s that has J as its jingle.

Input

The input file contains multiple test cases.

The first line of the input file contains t — the number of test cases.

Each of the following t lines contains a string s that consists of at most 100 000 lowercase characters.

The sum of lengths of strings for all test cases in the input file doesn't exceed 100 000.

Output

For each test case first output two integers: d — the number of different sets that appear as jingles of some substrings of s and $v(s)$ — the sum described in the problem statement.

Examples

jingles.in	jingles.out
2 abacaba abbcccdddd	6 36 10 125