Gondola

Mao-Kong Gondola is a famous attraction in Taipei. The gondola system consists of a circular rail, a single station, and n gondolas numbered consecutively from 1 to n running around the rail in a fixed direction. After gondola i passes the station, the next gondola to pass the station will be gondola $i + 1$ if $i < n$, or gondola 1 if $i = n$.

Gondolas may break down. Luckily we have an infinite supply of spare gondolas, which are numbered $n + 1$, $n + 2$, and so on. When a gondola breaks down we replace it (in the same position on the track) with the first available spare gondola, that is, the one with the lowest number. For example, if there are five gondolas and gondola 1 breaks down, then we will replace it with gondola 6.

You like to stand at the station and watch the gondolas as they pass by. A *gondola sequence* is a sequence of n numbers of gondolas that pass the station. It is possible that one or more gondolas broke down (and were replaced) before you arrived, but none of the gondolas break down while you are watching.

Note that the same configuration of gondolas on the rail can give multiple gondola sequences, depending on which gondola passes first when you arrive at the station. For example, if none of the gondolas have broken down then both (2, 3, 4, 5, 1) and (4, 5, 1, 2, 3) are possible gondola sequences, but (4, 3, 2, 5, 1) is not (because the gondolas appear in the wrong order).

If gondola 1 breaks down, then we might now observe the gondola sequence (4, 5, 6, 2, 3). If gondola 4 breaks down next, we replace it with gondola 7 and we might observe the gondola sequence (6, 2, 3, 7, 5). If gondola 7 breaks down after this, we replace it with gondola 8 and we may now observe the gondola sequence (3, 8, 5, 6, 2).

broken gondola	new gondola	possible gondola sequence
1	6	(4, 5, 6, 2, 3)
4	7	(6, 2, 3, 7, 5)
7	8	(3, 8, 5, 6, 2)

A *replacement sequence* is a sequence consisting of the numbers of the gondolas that have broken down, in the order in which they break down. In the previous example the replacement sequence is (1, 4, 7). A replacement sequence r *produces* a gondola sequence g if, after gondolas break down according to the replacement sequence r, the gondola sequence g may be observed.

Gondola Sequence Checking

In the first three subtasks you must check whether an input sequence is a gondola sequence. See the table below for examples of sequences that are and are not gondola sequences. You need to implement a function `valid`.

- `valid(n, inputSeq)`
  - `n`: the length of the input sequence.
  - `inputSeq`: array of length n; `inputSeq[i]` is element i of the input sequence, for $0 \leq i \leq n-1$.
  - The function should return 1 if the input sequence is a gondola sequence, or 0 otherwise.

Subtasks 1, 2, 3

subtask	points	n	inputSeq
1	5	$n \leq 100$	has each number from 1 to n exactly once
2	5	$n \leq 100,000$	$1 \leq$ `inputSeq[i]` $\leq n$
3	10	$n \leq 100,000$	$1 \leq$ `inputSeq[i]` $\leq 250,000$

Examples

subtask	inputSeq	return value	note
1	(1, 2, 3, 4, 5, 6, 7)	1	
1	(3, 4, 5, 6, 1, 2)	1	
1	(1, 5, 3, 4, 2, 7, 6)	0	1 cannot appear just before 5
1	(4, 3, 2, 1)	0	4 cannot appear just before 3
2	(1, 2, 3, 4, 5, 6, 5)	0	two gondolas numbered 5
3	(2, 3, 4, 9, 6, 7, 1)	1	replacement sequence (5, 8)
3	(10, 4, 3, 11, 12)	0	4 cannot appear just before 3

Replacement Sequence

In the next three subtasks you must construct a possible replacement sequence that produces a given gondola sequence. Any such replacement sequence will be accepted. You need to implement a function `replacement`.

- `replacement(n, gondolaSeq, replacementSeq)`
  - `n` is the length of the gondola sequence.
  - `gondolaSeq`: array of length n; `gondolaSeq` is guaranteed to be a gondola sequence, and `gondolaSeq[i]` is element i of the sequence, for $0 \leq i \leq n-1$.
  - The function should return l, the length of the replacement sequence.
  - `replacementSeq`: array that is sufficiently large to store the replacement sequence; you should return your sequence by placing element i of your replacement sequence into

`replacementSeq[i]`, for $0 \leq i \leq l-1$.

Subtasks 4, 5, 6

subtask	points	n	`gondolaSeq`
4	5	$n \leq 100$	$1 \leq$ `gondolaSeq[i]` $\leq n+1$
5	10	$n \leq 1,000$	$1 \leq$ `gondolaSeq[i]` $\leq 5,000$
6	20	$n \leq 100,000$	$1 \leq$ `gondolaSeq[i]` $\leq 250,000$

Examples

subtask	`gondolaSeq`	return value	`replacementSeq`
4	(3, 1, 4)	1	(2)
4	(5, 1, 2, 3, 4)	0	( )
5	(2, 3, 4, 9, 6, 7, 1)	2	(5, 8)

Count Replacement Sequences

In the next four subtasks you must count the number of possible replacement sequences that produce a given sequence (which may or may not be a gondola sequence), modulo **1,000,000,009**. You need to implement a function `countReplacement`.

- `countReplacement(n, inputSeq)`
  - `n`: the length of the input sequence.
  - `inputSeq`: array of length n; `inputSeq[i]` is element i of the input sequence, for $0 \leq i \leq n-1$.
  - If the input sequence is a gondola sequence, then count the number of replacement sequences that produce this gondola sequence (which could be extremely large), *and return this number modulo **1,000,000,009***. If the input sequence is not a gondola sequence, the function should return 0. If the input sequence is a gondola sequence but no gondolas broke down, the function should return 1.

Subtasks 7, 8, 9, 10

subtask	points	n	`inputSeq`
7	5	$4 \leq n \leq 50$	$1 \leq$ `inputSeq[i]` $\leq n+3$
8	15	$4 \leq n \leq 50$	$1 \leq$ `inputSeq[i]` ≤ 100, and at least $n-3$ of the initial gondolas $1, \ldots, n$ did not break down.
9	15	$n \leq 100,000$	$1 \leq$ `inputSeq[i]` $\leq 250,000$
10	10	$n \leq 100,000$	$1 \leq$ `inputSeq[i]` $\leq 1,000,000,000$

Examples

subtask	inputSeq	return value	replacement sequence
7	(1, 2, 7, 6)	2	(3, 4, 5) or (4, 5, 3)
8	(2, 3, 4, 12, 6, 7, 1)	1	(5, 8, 9, 10, 11)
9	(4, 7, 4, 7)	0	`inputSeq` is not a gondola sequence
10	(3, 4)	2	(1, 2) or (2, 1)

Implementation details

You have to submit exactly one file, called `gondola.c`, `gondola.cpp` or `gondola.pas`. This file should implement all three subprograms described above (even if you only plan to solve some of the subtasks), using the following signatures. You also need to include a header file `gondola.h` for C/C++ implementation.

C/C++ programs

```
int valid(int n, int inputSeq[]);
int replacement(int n, int gondolaSeq[], int replacementSeq[]);
int countReplacement(int n, int inputSeq[]);
```

Pascal programs

```
function valid(n: longint; inputSeq: array of longint): integer;
function replacement(n: longint; gondolaSeq: array of longint;
var replacementSeq: array of longint): longint;
function countReplacement(n: longint; inputSeq: array of longint):
longint;
```

Sample grader

The sample grader reads the input in the following format:

- line 1: `T`, the subtask number your program intends to solve ($\mathbf{1 \leq T \leq 10}$).
- line 2: `n`, the length of the input sequence.
- line 3: If `T` is 4, 5, or 6, this line contains `gondolaSeq[0]`, ..., `gondolaSeq[n-1]`. Otherwise this line contains `inputSeq[0]`, ..., `inputSeq[n-1]`.

Friend

We build a social network from n people numbered 0, ... , $n-1$. Some pairs of people in the network will be friends. If person x becomes a friend of person y, then person y also becomes a friend of person x.

The people are added to the network in n stages, which are also numbered from 0 to $n-1$. Person i is added in stage i. In stage 0, person 0 is added as the only person of the network. In each of the next $n-1$ stages, a person is added to the network by a *host*, who may be any person already in the network. At stage i ($0 < i < n$), the host for that stage can add the incoming person i into the network by one of the following three protocols:

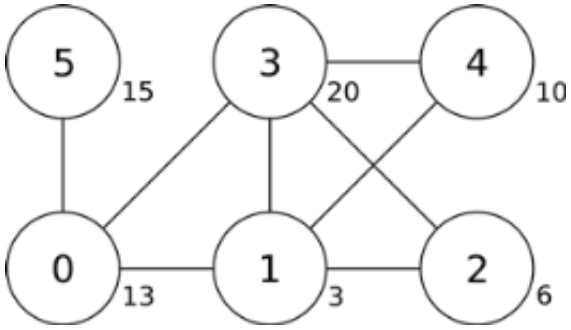
- *IAmYourFriend* makes person i a friend of the host only.
- *MyFriendsAreYourFriends* makes person i a friend of *each* person, who is a friend of the host at this moment. Note that this protocol does *not* make person i a friend of the host.
- *WeAreYourFriends* makes person i a friend of the host, and also a friend of *each* person, who is a friend of the host at this moment.

After we build the network we would like to pick a *sample* for a survey, that is, choose a group of people from the network. Since friends usually have similar interests, the sample should not include any pair of people who are friends with each other. Each person has a survey *confidence*, expressed as a positive integer, and we would like to find a sample with the maximum total confidence.

Example

stage	host	protocol	friend relations added
1	0	IAmYourFriend	(1, 0)
2	0	MyFriendsAreYourFriends	(2, 1)
3	1	WeAreYourFriends	(3, 1), (3, 0), (3, 2)
4	2	MyFriendsAreYourFriends	(4, 1), (4, 3)
5	0	IAmYourFriend	(5, 0)

Initially the network contains only person 0. The host of stage 1 (person 0) invites the new person 1 through the IAmYourFriend protocol, hence they become friends. The host of stage 2 (person 0 again) invites person 2 by MyFriendsAreYourFriends, which makes person 1 (the only friend of the host) the only friend of person 2. The host of stage 3 (person 1) adds person 3 through WeAreYourFriends, which makes person 3 a friend of person 1 (the host) and people 0 and 2 (the friends of the host). Stages 4 and 5 are also shown in the table above. The final network is shown in the following figure, in which the numbers inside the circles show the labels of people, and the numbers next to the circles show the survey confidence. The sample consisting of people 3 and 5 has total survey confidence equal to 20 + 15 = 35, which is the maximum possible total confidence.



Task

Given the description of each stage and the confidence value of each person, find a sample with the maximum total confidence. You only need to implement the function `findSample`.

- `findSample(n, confidence, host, protocol)`
  - `n`: the number of people.
  - `confidence`: array of length n; `confidence[i]` gives the confidence value of person i.
  - `host`: array of length n; `host[i]` gives the host of stage i.
  - `protocol`: array of length n; `protocol[i]` gives the protocol code used in stage i ($0 < i < n$): 0 for IAmYourFriend, 1 for MyFriendsAreYourFriends, and 2 for WeAreYourFriends.
  - Since there is no host in stage 0, `host[0]` and `protocol[0]` are undefined and should not be accessed by your program.
  - The function should return the maximum possible total confidence of a sample.

Subtasks

Some subtasks use only a subset of protocols, as shown in the following table.

subtask	points	n	confidence	protocols used
1	11	$2 \le n \le 10$	$1 \le$ confidence $\le 1,000,000$	All three protocols
2	8	$2 \le n \le 1,000$	$1 \le$ confidence $\le 1,000,000$	Only MyFriendsAreYourFriends
3	8	$2 \le n \le 1,000$	$1 \le$ confidence $\le 1,000,000$	Only WeAreYourFriends
4	19	$2 \le n \le 1,000$	$1 \le$ confidence $\le 1,000,000$	Only IAmYourFriend
5	23	$2 \le n \le 1,000$	All confidence values are 1	Both MyFriendsAreYourFriends and IAmYourFriend
6	31	$2 \le n \le 100,000$	$1 \le$ confidence $\le 10,000$	All three protocols

Implementation details

You have to submit exactly one file, called `friend.c`, `friend.cpp` or `friend.pas`. This file should implement the subprogram described above, using the following signatures. You also need to include a header file `friend.h` for C/C++ implementation.

C/C++ program

```
int findSample(int n, int confidence[], int host[], int protocol[]);
```

Pascal programs

```
function findSample(n: longint, confidence: array of longint, host: array
of longint; protocol: array of longint): longint;
```

Sample grader

The sample grader reads the input in the following format:

- line 1: `n`
- line 2: `confidence[0]`, ..., `confidence[n-1]`
- line 3: `host[1]`, `protocol[1]`, `host[2]`, `protocol[2]`, ..., `host[n-1]`, `protocol[n-1]`

The sample grader will print the return value of `findSample`.

Holiday

Jian-Jia is planning his next holiday in Taiwan. During his holiday, Jian-Jia moves from city to city and visits attractions in the cities.

There are n cities in Taiwan, all located along a single highway. The cities are numbered consecutively from 0 to $n-1$. For city i, where $0 < i < n-1$, the adjacent cities are $i-1$ and $i+1$. The only city adjacent to city 0 is city 1, and the only city adjacent to city $n-1$ is city $n-2$.

Each city contains some number of attractions. Jian-Jia has d days of holiday and plans to visit as many attractions as possible. Jian-Jia has already selected a city in which to start his holiday. In each day of his holiday Jian-Jia can either move to an adjacent city, or else visit all the attractions of the city he is staying, but not both. Jian-Jia will *never visit the attractions in the same city twice* even if he stays in the city multiple times. Please help Jian-Jia plan his holiday so that he visits as many different attractions as possible.

Example

Suppose Jian-Jia has 7 days of holiday, there are 5 cities (listed in the table below), and he starts from city 2. On the first day Jian-Jia visits the 20 attractions in city 2. On the second day Jian-Jia moves from city 2 to city 3, and on the third day visits the 30 attractions in city 3. Jian-Jia then spends the next three days moving from city 3 to city 0, and visits the 10 attractions in city 0 on the seventh day. The total number of attractions Jian-Jia visits is 20 + 30 + 10 = 60, which is the maximum number of attractions Jian-Jia can visit in 7 days when he starts from city 2.

city	number of attractions
0	10
1	2
2	20
3	30
4	1

day	action
1	visit the attractions in city 2
2	move from city 2 to city 3
3	visit the attractions in city 3
4	move from city 3 to city 2
5	move from city 2 to city 1
6	move from city 1 to city 0
7	visit the attractions in city 0

Task

Please implement a function `findMaxAttraction` that computes the maximum number of attractions Jian-Jia can visit.

- `findMaxAttraction(n, start, d, attraction)`
  - `n`: the number of cities.
  - `start`: the index of the starting city.
  - `d`: the number of days.
  - `attraction`: array of length n; `attraction[i]` is the number of attractions in city i, for $0 \leq i \leq n-1$.
  - The function should return the maximum number of attractions Jian-Jia can visit.

Subtasks

In all subtasks $0 \leq d \leq 2n + \lfloor n/2 \rfloor$, and the number of attractions in each city is nonnegative.

Additional constraints:

subtask	points	n	maximum number of attractions in a city	starting city
1	7	$2 \leq n \leq 20$	1,000,000,000	no constraints
2	23	$2 \leq n \leq 100,000$	100	city 0
3	17	$2 \leq n \leq 3,000$	1,000,000,000	no constraints
4	53	$2 \leq n \leq 100,000$	1,000,000,000	no constraints

Implementation details

You have to submit exactly one file, called `holiday.c`, `holiday.cpp` or `holiday.pas`. This file should implement the subprogram described above using the following signatures. You also need to include a header file `holiday.h` for C/C++ implementation.

Note that the result may be large, and the return type of `findMaxAttraction` is a 64-bit integer.

C/C++ program

```
long long int findMaxAttraction(int n, int start, int d,
int attraction[]);
```

Pascal program

```
function findMaxAttraction(n, start, d : longint;
attraction : array of longint): int64;
```

Sample grader

The sample grader reads the input in the following format:

- line 1: `n`, `start`, `d`.
- line 2: `attraction[0]`, ..., `attraction[n-1]`.

The sample grader will print the return value of `findMaxAttraction`.