Технические детали

- Вам необходимо посылать один файл (имя файла указано в условии задачи).
- Посылаемое решение должно реализовывать процедуры описанные в условии задачи, с описанными прототипами.
- Эти процедуры должны выполнять действия, описанные в условии задачи.
- Вы можете реализовать дополнительные процедуры для внутреннего использования.
- Ваше решение не должно взаимодействовать ни со стандартными потоками ввода-вывода, ни с другими файлами.

Ограничения по времени и памяти

Задача	Ограничение по времени	Ограничение по памяти
gondola	1 секунда	256 MB
friend	1 секунда	16 MB
holiday	5 секунд	64 MB

Гондола

Гондола Мао-Конга — известная достопримечательность Тайбея. Это подвесная дорога в виде окружности, которая имеет одну станцию и n гондол, пронумерованных от 1 до n. Гондолы движутся по этой дороге в одном направлении. Изначально, после гондолы с номером i мимо станции проезжает гондола с номером $(i+1)$ для $i < n$. Если $i = n$, то следующей проезжает гондола с номером 1.

Иногда гондолы могут ломаться. К счастью, у нас есть неограниченное количество запасных гондол, которые имеют номера $(n+1)$, $(n+2)$ и так далее. Когда гондола ломается, ее заменяют запасной с наименьшим номером среди тех, которые еще не были использованы. При этом новая гондола устанавливается на то же место, где стояла сломавшаяся. Например, если было пять гондол, и одна из них сломалась, то она будет заменена на гондолу с номером 6.

Вам нравится стоять на станции и смотреть на проезжающие мимо гондолы. *Последовательностью гондол* называется последовательность из n номеров гондол в порядке, в котором они проезжают мимо станции. Возможно, что одна или несколько гондол сломались до того, как вы пришли на станцию, но пока вы смотрите на движение гондол ни одна из гондол не ломается.

Обратите внимание, что одному и тому же порядку гондол может соответствовать несколько последовательностей, в зависимости от того, какая из гондол приехала на станцию первой. Например, если ни одна из гондол не ломалась, то возможные последовательности гондол (2, 3, 4, 5, 1) и (4, 5, 1, 2, 3), а (4, 3, 2, 5, 1) — не является последовательностью гондол, потому что гондолы приехали в неправильном порядке.

Если гондола с номером 1 сломается, то Вы сможете увидеть последовательность гондол (4, 5, 6, 2, 3). Если после этого сломается гондола с номером 4, ее заменят на гондолу с номером 7, и вы сможете увидеть последовательность (6, 2, 3, 7, 5). Если после этого гондола с номером 7 сломается, ее заменят на гондолу с номером 8, и вы сможете увидеть последовательность (3, 8, 5, 6, 2).

Сломанная гондола	Новая гондола	Одна из последовательностей гондол
1	6	(4, 5, 6, 2, 3)
4	7	(6, 2, 3, 7, 5)
7	8	(3, 8, 5, 6, 2)

Последовательностью замен называется последовательность номеров сломавшихся гондол в том порядке, в котором они ломались. В предыдущем примере последовательность замен — (1, 4, 7). Будем считать, что последовательность замен r *порождает* последовательность гондол g, если после того, как гондолы ломались и заменялись в соответствии с последовательностью замен r, вы можете увидеть последовательность гондол g.

Проверка последовательности гондол

В первых трех подзадачах вам необходимо проверить, является ли последовательность гондол корректной. Вам необходимо реализовать функцию `valid`.

- `valid(n, inputSeq)`
  - `n`: длина последовательности;
  - `inputSeq`: массив длины n; `inputSeq[i]` элемент последовательности на месте i, для $0 \leq i \leq n-1$;
  - функция должна возвращать 1, если переданная последовательность является последовательностью гондол, и 0 — в противном случае.

Подзадачи 1, 2, 3

Подзадача	Баллы	n	`inputSeq`
1	5	$n \leq 100$	каждое число от 1 до n встречается ровно один раз
2	5	$n \leq 100\,000$	$1 \leq$ `inputSeq[i]` $\leq n$
3	10	$n \leq 100\,000$	$1 \leq$ `inputSeq[i]` $\leq 250\,000$

Примеры

В таблице ниже приведены примеры последовательностей, которые являются и не являются последовательностями гондол.

Подзадачи	`inputSeq`	Возвращаемое значение	Комментарий
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 не может быть перед 5
1	(4, 3, 2, 1)	0	4 не может быть перед 3
2	(1, 2, 3, 4, 5, 6, 5)	0	две гондолы с номером 5
3	(2, 3, 4, 9, 6, 7, 1)	1	последовательность замен (5, 8)
3	(10, 4, 3, 11, 12)	0	4 не может быть перед 3

Последовательность замен

В следующих трех подзадачах вы должны построить последовательность замен, которая порождает заданную последовательность гондол. Вы можете построить любую из таких последовательностей.

Вам необходимо реализовать функцию `replacement`.

- `replacement(n, gondolaSeq, replacementSeq)`
  - `n`: длина последовательности;
  - `gondolaSeq`: массив длины n; гарантируется, что `gondolaSeq` является последовательностью гондол; `gondolaSeq[i]` — это i-ый элемент последовательности, для $0 \leq i \leq n-1$;
  - функция должна возвращать l — длину последовательности замен;
  - `replacementSeq`: массив, достаточно большой, чтобы вместить последовательность замен; вы должны вернуть последовательность замен, записав ее i-ый элемент в `replacementSeq[i]`, для всех $0 \leq i \leq l-1$.

Подзадача 4, 5, 6

Подзадача	Баллы	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]` $\leq 250\,000$

Примеры

Подзадача	`gondolaSeq`	Возвращаемое значение	`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)

Подсчет последовательностей замен

В следующих четырех подзадачах вы должны определить количество последовательностей замен, которые порождают заданную последовательность (которая может быть последовательностью гондол, а может не быть) по модулю $1\,000\,000\,009$. Вы должны реализовать функцию `countReplacement`.

- `countReplacement(n, inputSeq)`
  - `n`: длина переданной последовательности;
  - `inputSeq`: массив длины n; `inputSeq[i]` — это i-ый элемент переданной последовательности, для всех $0 \leq i \leq (n-1)$;
  - если переданная последовательность является последовательностью гондол, вы должны определить количество последовательностей замен, которые порождают данную последовательность гондол, и *вернуть остаток от деления этого количества на* $1\,000\,000\,009$. Если переданная последовательность не является последовательностью гондол, функция должна возвращать 0. Если переданная последовательность является последовательностью гондол, но ни одна гондола не сломалась, функция должна вернуть 1.

Подзадачи 7, 8, 9, 10

Подзадача	Баллы	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, хотя бы $(n-3)$ из первых гондол $(1, \ldots, n)$ не сломались
9	15	$n \leq 100\,000$	$1 \leq$ `inputSeq[i]` $\leq 250\,000$
10	10	$n \leq 100\,000$	$1 \leq$ `inputSeq[i]` $\leq 1\,000\,000\,000$

Примеры

Подзадача	`inputSeq`	Возвращаемое значение	Последовательность замен
7	(1, 2, 7, 6)	2	(3, 4, 5) или (4, 5, 3)
8	(2, 3, 4, 12, 6, 7, 1)	1	(5, 8, 9, 10, 11)
9	(4, 7, 4, 7)	0	`inputSeq` не является последовательностью гондол
10	(3, 4)	2	(1, 2) или (2, 1)

Детали реализации

Вы должны послать ровно один файл, названный `gondola.c`, `gondola.cpp` или `gondola.pas`. В этом файле должны быть реализованы функции, описанные выше с указанными ниже прототипами. Все три функции должны быть реализованы, даже если вы не планируете решать все подзадачи. На языках C/C++ вы должны подключить заголовочный файл `gondola.h`.

Язык C/C++

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

Язык Pascal

```
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;
```

Пример проверяющего модуля

Предоставленный пример проверяющего модуля имеет следующий формат входных данных:

- строка 1: `T`, номер подзадачи, которую должна решать ваша программа ($1 \leq T \leq 10$);
- строка 2: `n`, длина переданной последовательности;
- строка 3: если `T` равно 4, 5 или 6, эта строка содержит `gondolaSeq[0]`, ..., `gondolaSeq[n-1]`. Иначе она содержит `inputSeq[0]`, ..., `inputSeq[n-1]`.

Друзья

Создается социальная сеть, состоящая из n участников, пронумерованных 0, ... , $(n-1)$. Некоторые пары участников этой сети могут стать друзьями. Если участник x становится другом участника y, то участник y также становится другом участника x.

Участники добавляются в сеть за n этапов, которые также пронумерованы от 0 до $(n-1)$. Участник i добавляется на этапе i. На этапе 0 добавляется участник с номером 0 как единственный участник сети. На каждом из следующих $(n-1)$ этапов очередной участник добавляется в сеть *хозяином* этапа, которым может быть любой участник, уже добавленный в сеть. На этапе i ($0 < i < n$), хозяин этапа может добавить очередного участника i в сеть по одному из трех протоколов:

- *IAmYourFriend* делает участника i другом только хозяина этапа.
- *MyFriendsAreYourFriends* делает участника i другом *каждого* друга хозяина в этот момент. Заметьте, что этот протокол *не* делает участника i другом хозяина.
- *WeAreYourFriends* делает участника i другом хозяина в этот момент, а также другом *каждого* друга хозяина.

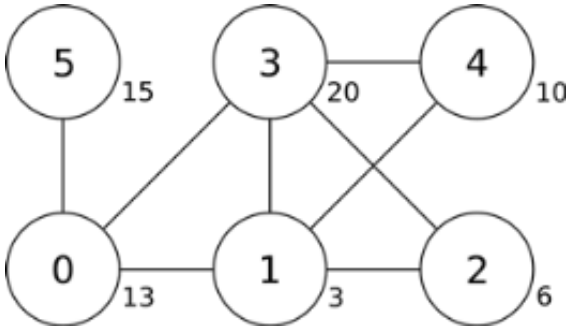
После того, как сеть создана, необходимо сделать *выборку* для опроса, то есть отобрать группу участников сети. Поскольку друзья обычно имеют общие интересы, эта выборка не должна содержать пары участников, являющихся друзьями. Каждый участник имеет некоторый *уровень доверия* в опросах, который задан положительным целым числом, и нужно сделать выборку участников с максимальным суммарным уровнем доверия.

Пример

Этап	Хозяин	Протокол	Добавленные пары друзей
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)

Вначале сеть содержит только участника с номером 0. Хозяин первого этапа (участник с номером 0) приглашает нового участника с номером 1, используя протокол IAmYourFriend, и они становятся друзьями. Хозяин второго этапа (снова участник с номером 0) приглашает второго участника, используя протокол MyFriendsAreYourFriends, который делает участника с номером 1 (единственный друг хозяина) единственным другом участника с номером 2. Хозяин третьего этапа (участник с номером 1) добавляет третьего участника, используя протокол WeAreYourFriends, что делает третьего участника другом первого участника (хозяина) и

участников с номерами 0 и 2 (они же друзья хозяина). Этапы 4 и 5 также показаны в таблице выше. Финальная сеть представлена на рисунке ниже, где число в кружке показывает номер участника, а число рядом с кружком показывает уровень доверия в опросах для этого участника. Выборка, состоящая из участников с номерами 3 и 5, имеет суммарный уровень доверия в опросах, который составляет 20 + 15 = 35, что является максимально возможным суммарным уровнем доверия.



Постановка задачи

Имея описание каждого этапа и уровень доверия в опросах каждого участника, необходимо найти выборку участников сети с максимальным суммарным уровнем доверия. Вы должны реализовать функцию `findSample`.

- `findSample(n, confidence, host, protocol)`
  - `n`: количество участников;
  - `confidence`: массив длины n; `confidence[i]` задает уровень доверия к участнику с номером i;
  - `host`: массив длины n; `host[i]` задает хозяина i-го этапа;
  - `protocol`: массив длины n; `protocol[i]` задает код протокола, используемого на i-ом этапе ($0 < i < n$): 0 — для IAmYourFriend, 1 — для MyFriendsAreYourFriends, 2 — для WeAreYourFriends;
  - поскольку на этапе 0 нет хозяина, и `host[0]` и `protocol[0]` не определены, то ваша программа не должна к ним обращаться;
  - функция должна возвращать максимально возможный суммарный уровень доверия для выборки участников.

Подзадачи

Некоторые подзадачи используют не все протоколы, как показано в таблице ниже.

Подзадача	Баллы	n	Уровень доверия (confidence)	Используемые протоколы
1	11	$2 \leq n \leq 10$	$1 \leq \text{confidence} \leq 1\,000\,000$	Все три протокола
2	8	$2 \leq n \leq 1\,000$	$1 \leq \text{confidence} \leq 1\,000\,000$	Только MyFriendsAreYourFriends
3	8	$2 \leq n \leq 1\,000$	$1 \leq \text{confidence} \leq 1\,000\,000$	Только WeAreYourFriends
4	19	$2 \leq n \leq 1\,000$	$1 \leq \text{confidence} \leq 1\,000\,000$	Только IAmYourFriend
5	23	$2 \leq n \leq 1\,000$	Все уровни доверия равны 1	Только MyFriendsAreYourFriends и IAmYourFriend
6	31	$2 \leq n \leq 100\,000$	$1 \leq \text{confidence} \leq 10\,000$	Все три протокола

Детали реализации

Вы должны послать ровно один файл, названный `friend.c`, `friend.cpp` или `friend.pas`. В этом файле должна быть реализована функция, описанная выше с указанными ниже прототипами. На языках C/C++ вы должны подключить заголовочный файл `friend.h`.

Язык C/C++

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

Язык Pascal

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

Пример проверяющего модуля

Предоставленный пример проверяющего модуля имеет следующий формат входных данных:

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

Предоставленный пример проверяющего модуля выведет значение, возвращаемое функцией `findSample`.

Отпуск

Джан-Джи планирует провести свой следующий отпуск в Тайване. Во время отпуска он собирается переезжать из города в город и посещать достопримечательности в этих городах.

В Тайване n городов, которые расположены вдоль единственной магистрали. Города пронумерованы последовательно целыми числами от 0 до $(n-1)$. Для i-ого города, $0 < i < n-1$, соседними являются города с номерами $(i-1)$ и $(i+1)$. Город с номером 0 соседствует только с городом с номером 1, а город с номером $(n-1)$ соседствует только с городом с номером $(n-2)$.

В каждом городе содержится некоторое количество достопримечательностей. Джан-Джи планирует посетить как можно больше достопримечательностей во время своего отпуска продолжительностью d дней. Он уже выбрал город, с которого начнет отпуск. Каждый день Джан-Джи может либо переехать из текущего города в один из соседних, либо посетить все достопримечательности в городе, в котором он находится. Он не может сделать оба действия в один день. Джан-Джи *никогда не посещает достопримечательность дважды в одном городе*, даже если приезжает в этот город несколько раз. Пожалуйста, помогите Джан-Джи спланировать отпуск так, чтобы он посетил как можно больше достопримечательностей.

Пример

Пусть отпуск Джан-Джи длится 7 дней, количество городов равно 5 (города описаны в таблице ниже), и он начинает свой отпуск в городе с номером 2. В первый день Джан-Джи посещает 20 достопримечательностей в городе с номером 2. Во второй день Джан-Джи переезжает из города с номером 2 в город с номером 3, и в третий день он посещает 30 достопримечательностей в городе с номером 3. Следующие три дня Джан-Джи тратит на переезд из города с номером 3 в город с номером 0 и в седьмой день посещает 10 достопримечательностей в городе с номером 0. Общее количество достопримечательностей, которые посетил Джан-Джи, составляет 20 + 30 + 10 = 60, что является максимальным количеством достопримечательностей, которые он может посетить за 7 дней, начав отпуск в городе с номером 2.

Город	Количество достопримечательностей
0	10
1	2
2	20
3	30
4	1

День	Действие
1	посещение достопримечательностей в городе с номером 2
2	переезд из города с номером 2 в город с номером 3
3	посещение достопримечательностей в городе с номером 3
4	переезд из города с номером 3 в город с номером 2
5	переезд из города с номером 2 в город с номером 1
6	переезд из города с номером 1 в город с номером 0
7	посещение достопримечательностей в городе с номером 0

Постановка задачи

Вам требуется реализовать функцию `findMaxAttraction`, которая вычисляет максимальное количество достопримечательностей, которые Джан-Джи может посетить.

- `findMaxAttraction(n, start, d, attraction)`
  - `n`: количество городов;
  - `start`: номер начального города;
  - `d`: количество дней;
  - `attraction`: массив длины n; `attraction[i]` задает количество достопримечательностей в i-ом городе;
  - функция должна возвращать максимальное количество достопримечательностей, которые Джан-Джи может посетить.

Подзадачи

Во всех подзадачах выполнено условие $0 \le d \le 2n + \lfloor \frac{n}{2} \rfloor$, во всех городах количество достопримечательностей неотрицательно.

Подзадача	Баллы	n	Максимальное количество достопримечательностей в городе	Начальный город
1	7	$2 \le n \le 20$	1 000 000 000	нет дополнительных ограничений
2	23	$2 \le n \le 100\,000$	100	город 0
3	17	$2 \le n \le 3\,000$	1 000 000 000	нет дополнительных ограничений
4	53	$2 \le n \le 100\,000$	1 000 000 000	нет дополнительных ограничений

Детали реализации

Вы должны послать ровно один файл, названный `holiday.c`, `holiday.cpp` или `holiday.pas`. В этом файле должна быть реализована функция, описанная выше с указанными ниже прототипами. На языках C/C++ вы должны подключить заголовочный файл `holiday.h`.

Обратите внимание, что результат может быть достаточно большим, и функция `findMaxAttraction` возвращает 64-битное целое число.

Язык C/C++

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

Язык Pascal

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

Пример проверяющего модуля

Предоставленный пример проверяющего модуля имеет следующий формат входных данных:

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

Предоставленный пример проверяющего модуля напечатает значение, возвращаемое функцией `findMaxAttraction`.