Кеширующий сервер

Имя входного файла:	`стандартный ввод`
Имя выходного файла:	`стандартный вывод`
Ограничение по времени:	1 секунда
Ограничение по памяти:	64 мегабайта

Дормидонт работает в компании, которая занимается обработкой больших данных. Обрабатываемые данные находятся где-то в распределённой системе. Количество различных данных в системе ограничено и каждое данное имеет свой номер. Эти данные регулярно требуются различным клиентам и, поскольку время обращения к ним достаточно велико, для ускорения обработки информации Дормидонту поручено написать часть middlware — сервер-посредник, к которому и обращаются теперь клиенты за данными. Так как система — распределённая, а сервер — нет, все требуемые данные на сервер не помещаются, но он имеет возможность запоминать результаты своих запросов к распределённой системе. Для этого на сервере выделена ограниченная память на N запросов.

К большой радости Дормидонта оказалось, что самые крупные и значимые клиенты всегда обращаются за одними и теми же данными в одном и том же порядке, так что у него есть последовательность запросов. Дормидонт придумал такой алгоритм, что как можно большее количество запросов исполняется из кеша сервера, без обращения к распределённой системе. Придумаете ли вы что-то подобное?

Формат входных данных

На вход программы подаётся размер памяти под кеширование запросов $1 \leqslant N \leqslant 10000$, количество запросов $1 \leqslant M \leqslant 10000$ и ровно M запросов с номерами $0 \leqslant R_i \leqslant 10^9$.

Формат выходных данных

Требуется вывести одно число: сколько раз пришлось обратиться к распределённой системе за данными, отсутствующими в кеше. В начале работы кеш пуст.

Пример

стандартный ввод	стандартный вывод
5 15 3 1 4 1 5 9 2 6 5 3 5 8 7 9 3	9

Замечание

В приведённом примере первые три запроса произойдут к данным под номерами 3, 1 и 4, так как их нет в кеше. Следующий запрос, 1, есть в кеше и обращения к распределённой системе не произойдёт. Запросы 5 и 9 занесут их в кеш. Следующий запрос — 2 — в кеше отсутствует, но мы выкинем из кеша запрос 1 и запрос 2 займёт его место. Далее, запрос 6 вытеснит из кеша значение 2, после чего следующие три запроса удовлетворятся из кеша. Затем произойдёт ещё два вытеснения — 8 и 7. Итого 9 обращений к распределённой системе. Нетрудно установить, что меньше сделать нельзя.