Задача 1. Удаление повторов

Источник:	базовая*
Имя входного файла:	`input.txt`
Имя выходного файла:	`output.txt`
Ограничение по времени:	1 секунда
Ограничение по памяти:	разумное

Из списка, элементами которого являются целые числа, убрать повторения (т.е. подряд не должны идти одинаковые элементы.)

Формат входных данных

Во входном файле через пробел записаны целые числа. Из этих чисел нужно построить односвязный список в порядке их встречаемости во входном файле. Количество чисел от 1 до 10^5.

Формат выходных данных

Из полученного списка нужно удалить все повторяющиеся элементы, а затем в выходной файл выдать через пробел оставшиеся числа. Память освободить.

Пример

input.txt	output.txt
1 4 5 6 6 4 2 3 3 3	1 4 5 6 4 2 3

Задача 2. Удаление предшественников

Источник:	базовая
Имя входного файла:	`input.txt`
Имя выходного файла:	`output.txt`
Ограничение по времени:	1 секунда
Ограничение по памяти:	разумное

По заданной последовательности целых чисел построить односвязный список. Каждое новое число добавлять в начало списка. Затем пройти по построенному списку и удалить элемент, предшествующий элементу, содержащему заданное число, для всех таких вхождений. После этого распечатать полученный список, память освободить.

Формат входных данных

Первая строка входного файла содержит заданное число, которое нужно удалить.

В следующей строке записана последовательность целых чисел. Числа в файле записаны через пробел. Их величина по модулю не превосходит 1000. Количество чисел может изменяться от 1 до 1000.

Формат выходных данных

В выходной файл нужно вывести последовательность элементов, оставшихся в списке после удаления.

Пример

input.txt	output.txt
4 1 5 4 6 4 3	4 4 5 1

Задача 3. Двусвязный список

Источник:	базовая
Имя входного файла:	`input.txt`
Имя выходного файла:	`output.txt`
Ограничение по времени:	1 секунда
Ограничение по памяти:	разумное

Требуется реализовать двусвязный список. В каждом узле списка хранится целочисленное значение и индексы следующего и предыдущего узлов. Дано начальное состояние списка, а также последовательность операций двух видов: добавление и удаление узла. Нужно выполнить все операции, и после этого вывести значения всех узлов списка в порядке их следования в цепи.

У каждого узла списка есть индекс в едином массиве, в котором всё хранится. В начальном состоянии списка есть N узлов, имеющих индексы от 0 до $N-1$ в порядке их задания. При добавлении нового узла он дописывается в конец массива. То есть первый добавленный узел имеет индекс N, второй добавленный — $N+1$, и так далее.

Нужно выполнять операции двух видов:

1. *Добавление узла спереди.* При этом указывается индекс узла, после которого нужно вставить новый узел, и значение для нового узла. Если индекс равен -1, то узел надо вставить в начало списка (перед самым первым элементом). После выполнения операции нужно вывести индекс нового узла (они назначаются по порядку, см. выше).
-1. *Добавление узла сзади.* При этом указывается индекс узла, перед которым нужно вставить новый узел, и значение для нового узла. Если индекс равен -1, то узел надо вставить в конец списка (после самого последнего элемента). После выполнения операции нужно вывести индекс нового узла.
0. *Удаление узла.* При этом указывается индекс узла, который нужно удалить. После выполнения операции нужно вывести значение удалённого узла.

В задаче используется “мультитест”: в одном входном файле записано много отдельных тестов.

Подсказка: Возможно, удобнее будет хранить связный список в виде двунаправленного кольцевого списка со вспомогательным элементом, например, сдвинув нумерацию узлов в программе на один.

Формат входных данных

В первой строке файла записано одно целое число T — количество тестов в файле. Далее в файле идут тесты (T штук) подряд, один за другим.

Первая строка теста начинается с целых чисел: N — изначальное количество узлов в связном списке, F — индекс первого элемента списка, L — индекс последнего элемента списка и Q — количество операций, которые нужно выполнить ($0 \leqslant F, L < N \leqslant 10^5$, $0 \leqslant Q \leqslant 10^5$).

Затем идёт N строк, в которых описываются узлы связного списка в порядке увеличения индекса. Описание каждого узла состоит из его целочисленного значения и двух индексов: следующего узла в списке и предыдущего (или -1, если одного из них нет).

Наконец, в тесте идут Q строк, которые описывают операции над списком. В каждой строке сначала записан тип операции: 1 — добавление спереди, -1 — добавление сзади, 0 — удаление. Затем указан индекс узла. Если описывается операция вставки, то в конце также задано целочисленное значение нового узла.

Все значения узлов лежат в диапазоне от 0 до 10^6 включительно.

Сумма N по всем тестах не превышает 10^5, аналогично для суммы всех Q.

Формат выходных данных

Для каждого теста нужно вывести следующее:

1. результаты выполнения операций (Q строк)
2. строка "===" (три знака равенства)
3. целочисленные значения всех узлов списка после выполнения операций (в порядке их следования в списке)
4. снова строка "==="

Пример

input.txt	output.txt
3 5 3 2 0 4283 2 4 2718 4 3 5000 -1 0 1111 1 -1 3141 0 1 1 0 0 5 0 -1 -1 1 -1 1000 -1 -1 2000 1 1 3000 -1 0 4000 0 2 4 2 3 2 45 1 2 74 3 0 16 0 -1 56 -1 1 0 1 1 2 35	=== 1111 2718 3141 4283 5000 === 1 2 3 4 2000 === 1000 3000 4000 0 === 74 4 === 16 35 45 56 ===

Пояснение к примеру

В примере три теста, похожие на тесты из задачи “Односвязный список”. Во всех тестах заменены строковые значения на целочисленные, во втором тесте также другой порядок узлов.

Задача 4. Список с указателями

Источник:	основная*
Имя входного файла:	`input.txt`
Имя выходного файла:	`output.txt`
Ограничение по времени:	1 секунда
Ограничение по памяти:	разумное

Удобно было бы реализовать связный список один раз, а потом использовать его в разных задачах для хранения разных типов данных (`int`, `double`, `char*`, ...). Однако сделать это не получается из-за того, что тип значения должен быть прописан в структуре узла. Самый простой способ решения этой проблемы — хранить в узлах списка не само значение, а указатель на него (т.е. его адрес). Если хранить внутри узла указатель `void*`, тогда в него можно будет записать указатель на значение любого типа.

В данной задаче нужно реализовать набор функций для работы со связным списком, в котором всегда хранятся указатели. Список должен быть двусвязный со вспомогательным узлом. В узлах списка требуется хранить указатели на следующий или предыдущей узлы (уже `не` индексы, как можно было в предыдущих задачах). Каждый узел списка должен быть размещён в динамической памяти, выделенной с помощью `malloc`.

Нужно реализовать следующие функции:

```
typedef struct Node_s {
    struct Node_s *prev, *next;
    void *value;
} Node;
//List -- вспомогательный узел, являющийся головой списка
typedef Node List;

//инициализирует поля структуры *list значениями для пустого списка
void initList(List *list);
//создаёт новый узел со значением ptr и вставляет его после узла node
//возвращает указатель на созданный узел
Node *addAfter(Node *node, void *ptr);
//создаёт новый узел со значением ptr и вставляет его перед узлом node
//возвращает указатель на созданный узел
Node *addBefore(Node *node, void *ptr);
//удаляет заданный узел, возвращая значение, которое в нём лежало
void *erase(Node *node);
```

Эти функции должны выделять/удалять память для узлов списка `Node`, но **не** для значений: вызывающий полностью контролирует то, куда указывают значения и как для них используется память.

Используя эти функции, решите тестовую задачу, аналогичную задаче “Двусвязный список”.

Формат входных данных

В первой строке файла записано одно целое число T — количество тестов в файле. Далее в файле идут тесты (T штук) подряд, один за другим.

Первая строка теста начинается с целого числа Q — количество операций, которые нужно выполнить ($0 \leqslant Q \leqslant 10^5$). В отличие от задачи “Двусвязный список”, в данной задаче полагается, что в начале теста список пустой.

Затем идут Q строк, которые описывают операции над списком. В каждой строке сначала записан тип операции: 1 — добавление спереди, -1 — добавление сзади, 0 — удаление. Затем указан индекс узла. Если описывается операция вставки, то в конце также задано целочисленное значение нового узла.

Все значения узлов лежат в диапазоне от 0 до 10^6 включительно.

Сумма Q по всем тестах не превышает 10^5.

Формат выходных данных

Для каждого теста нужно вывести значения всех узлов списка после выполнения операций (в порядке их следования в списке), и строку "===" в конце.

Пример

input.txt	output.txt
2 5 1 -1 4283 -1 0 2718 1 0 5000 -1 1 1111 1 1 3141 6 1 -1 0 1 -1 1000 -1 -1 2000 1 1 3000 -1 0 4000 0 2	1111 2718 3141 4283 5000 === 1000 3000 4000 0 ===

Пояснение к примеру

В примере два теста, похожие на тесты из задачи “Двусвязный список”. Различие лишь в том, что начальное состояние списка обеспечивается операциями, и третьего теста нет.

Задача 5. Задача Иосифа

Источник:	основная
Имя входного файла:	`input.txt`
Имя выходного файла:	`output.txt`
Ограничение по времени:	1 секунда
Ограничение по памяти:	разумное

Согласно легенде, еврейский историк Флавий Иосиф (37 н.э. – ок. 100 н.э.) был вовлечен в сражение между евреями и римлянами и прятался в пещере еще с 40 людьми. Пещера была окружена римскими воинами. Казалось неизбежным, что людей Иосифа обнаружат и захватят в плен. Для большинства из находившихся там смерть была более почетной, чем плен, и группа решила совершить самоубийство следующим образом: все становились в круг, начинали считать людей по кругу и убивали каждого третьего, пока не оставался один человек, который должен был убить себя сам. В отличие от остальных, Иосифу и его другу идея плена нравилась больше; они быстро (лихорадочно) вычислили, где им стать в круге, чтобы оказаться двумя последними.

Вам нужно решить более общую задачу: исключать из группы, первоначально состоящей из N элементов, каждого K-го до тех пор, пока в ней не останется один элемент. Для решения ее нужно использовать двусвязный циклический список.

Формат входных данных

Входной файл содержит два натуральных числа N и K – количество элементов в группе и номер каждого удаляемого ($1 \leqslant N, K \leqslant 1000$).

Из чисел от 1 до N построить двусвязный циклический список.

Формат выходных данных

Программа должна удалять из списка каждый K-й элемент, начинать счет нужно с первого элемента, а продолжать со следующего за удаленным.

В выходной файл нужно вывести одно целое число – номер элемента, который останется единственным неудаленным в списке.

Пример

input.txt	output.txt
10 3	4

Задача 6. Слияние списков

Источник:	основная
Имя входного файла:	`input.txt`
Имя выходного файла:	`output.txt`
Ограничение по времени:	1 секунда
Ограничение по памяти:	разумное

Дано два односвязных упорядоченных по неубыванию списка, содержащих целые числа. Нужно эти два списка слить в один, упорядоченный по невозрастанию, т.е. построить этот список из элементов двух данных. Новой памяти не выделять.

Формат входных данных

В первой строке входного файла записаны через пробел два положительных целых числа N и — длины первого и второго списков ($1 \leqslant N, M \leqslant 1000$).

В следующих двух строках через пробел записаны N и целых чисел соответственно – элементы первого и второго списков. В каждой строке числа даны в неубывающем порядке. Из них нужно построить два списка в том же порядке.

Формат выходных данных

Из двух списков нужно построить один, в котором элементы будут упорядочены по невозрастанию.

В выходной файл нужно вывести значения элементов полученного списка в одну строку через пробел. После этого память освободить.

Пример

input.txt	output.txt
4 5 1 1 7 9 2 3 7 8 9	9 9 8 7 7 3 2 1 1

Задача 7. Сортировка со списками

Источник:	основная*
Имя входного файла:	`input.txt`
Имя выходного файла:	`output.txt`
Ограничение по времени:	1 секунда
Ограничение по памяти:	разумное

В первой строке записано целое число N — количество записей ($1 \leqslant N \leqslant 2 \cdot 10^5$). В остальных N строках содержатся записи, по одной в строке.

Для каждой записи указаны ключ и значение через пробел. Ключ — это целое число в диапазоне от 0 до 10^6 включительно, а значение — это строка от одного до семи символов включительно, состоящая только из маленьких букв латинского алфавита.

Требуется вывести ровно те же самые N записей, но в другом порядке. Записи должны быть упорядочены по возрастанию ключа. Если у нескольких записей ключ равный, то нужно упорядочить их в том порядке, в котором они встречаются по входном файле.

Важно: Решать задачу **нужно** следующим образом (другие решения засчитываться **не** будут). Нужно завести 10^6 связных списков, и в каждый k-ый список складывать все записи с ключом, равным k. Тогда после раскидывания записей по спискам достаточно будет пробежаться по спискам в порядке увеличения k и распечатать их.

Пример

input.txt	output.txt
7 3 qwerty 3 string 6 good 1 a 3 ab 2 hello 5 world	1 a 2 hello 3 qwerty 3 string 3 ab 5 world 6 good

Пояснение к примеру

В примере 7 записей с ключами 1, 2, 3, 5 и 6 — именно в таком порядке записи и выведены в выходном файле. Обратите внимание, что есть три записи с ключом 3: `qwerty`, `string`, `ab`. Они выведены ровно в том порядке, в котором они идут во входном файле.

Задача 8. XOR-список

Источник:	повышенной сложности*
Имя входного файла:	`input.txt`
Имя выходного файла:	`output.txt`
Ограничение по времени:	1 секунда
Ограничение по памяти:	разумное

В узле односвязного списка хранится один указатель, но по такому списку нельзя перемещаться в обратную сторону и неудобно удалять элементы. В двусвязном списке всё просто и красиво, но приходится хранить уже по два указателя в каждом узле. Данная задача посвящена очень странной конструкции, когда в каждом узле хранится только одно дополнительное значение, но при этом можно перемещаться в обе стороны.

Как известно, указатель — это просто адрес в памяти, а над адресами можно выполнять арифметические операции. В узле предлагаемого списка вместо указателей `next` и `prev` нужно хранить, к примеру, сумму этих двух адресов. Тогда если в программе известны указатели на два подряд идущих элемента такого списка `left` и `right`, то адрес соседнего узла можно найти вычитанием адреса одного узла из хранящейся в другом узле суммы адресов соседей. Общий принцип заключается в том, что для выполнения любой операции (даже перемещения) в таком списке необходимо иметь два указателя, которые смотрят на два подряд идущих элемента списка, потому что по одному указателю невозможно найти соседние узлы.

Канонически, вместо суммы адресов соседей в узле обычно хранят “побитовое исключающее или” (т.е. XOR) адресов соседей. Тому есть несколько причин: во-первых, операция XOR играет роль сложения и вычитания одновременно; во-вторых, операция XOR порой выполняется немного быстрее сложения и вычитания, т.к. её проще реализовать на аппаратном уровне. Канонический вариант XOR-списка описан в википедии.

Требуется реализовать XOR-список и решить тестовую задачу. Примерно как в задаче “Список с указателями”, но теперь в узлах надо хранить целочисленные значения.

Нужно реализовать следующие функции:

```
#define VALTYPE int      //тип значений: целое
typedef struct Node_s {
    size_t xorlinks;     //XOR адресов соседних узлов
    VALTYPE value;       //значение в этом узле
} Node;
typedef struct List_s {
    ???                  //тут можно хранить всякие данные
} List;

//инициализировать пустой список в list
void initList(List *list);
//добавить новый узел со значением val между соседними узлами left и right
Node *addBetween(Node *left, Node *right, VALTYPE val);
//дано два соседних узла left и right, нужно удалить узел left
VALTYPE eraseLeft(Node *left, Node *right);
//дано два соседних узла left и right, нужно удалить узел right
VALTYPE eraseRight(Node *left, Node *right);
```

Для простоты реализации рекомендуется сделать XOR-список кольцевым с **двумя** вспомогательными элементами (“начало” и “конец”).

Формат входных данных

В первой строке файла записано одно целое число T — количество тестов в файле. Далее в файле идут тесты (T штук) подряд, один за другим.

Первая строка теста начинается с целого числа Q — количество операций, которые нужно выполнить ($0 \leqslant Q \leqslant 10^5$). В начале каждого теста список пустой.

Затем идут Q строк, которые описывают операции над списком. В каждой строке сначала записан тип операции: 1 — удаление правого, -1 — удаление левого, 0 — добавление. Затем указано два индекса: для узла `left` и узла `right`. Если описывается операция вставки, то в конце также задано целочисленное значение нового узла.

Гарантируется, что указанный узел `right` всегда идёт сразу после указанного узла `left`. Один из этих узлов может быть не указан (тогда записан индекс -1), если выполняется операция на краю списка.

Все значения узлов лежат в диапазоне от 0 до 10^6 включительно.

Сумма Q по всем тестам не превышает 10^5.

Формат выходных данных

Для каждого теста нужно вывести строковые значения всех узлов списка после выполнения операций (в порядке их следования в списке), и строку "===" в конце.

Пример

input.txt	output.txt
2 5 0 -1 -1 4283 0 -1 0 2718 0 0 -1 5000 0 -1 1 1111 0 1 0 3141 7 0 -1 -1 0 0 -1 0 1000 0 0 -1 2000 0 1 0 3000 0 2 -1 4000 1 0 2 -1 0 4	1111 2718 3141 4283 5000 === 1000 3000 4000 ===

Задача 9. Список с индексами

Источник:	повышенной сложности*
Имя входного файла:	`input.txt`
Имя выходного файла:	`output.txt`
Ограничение по времени:	1 секунда*
Ограничение по памяти:	разумное

У связного списка есть серьёзная проблема: в отличие от массива в нём нельзя быстро получить k-ый по порядку элемент. Это можно сделать лишь перебором узлов списка за $O(k) \approx O(N)$ времени. В данном задаче предлагается ускорить поиск узла по индексу.

Рассмотрим односвязный список. Пусть в списке помимо обычных узлов ("маленьких" узлов) есть ещё немного особенных узлов ("больших" узлов). В любом узле списка нужно хранить значение узла и указатель на следующий по порядку элемент. В каждом большом узле списка предлагается дополнительно хранить указатель на следующий **большой** узел списка и расстояние до него. Под расстоянием здесь понимается то, сколько переходов вперёд нужно сделать из одного узла, чтобы попасть по второй узел.

Дополнительные ссылки в больших узлах позволяют быстрее найти k-ый элемент, так как можно проскакивать сразу по много узлов, изначально проходя исключительно по большим узлам. Можно заметить, что если доля больших узлов в списке примерно $\frac{1}{B}$, то найти k-ый элемент можно примерно за $O(\frac{k}{B} + B)$. (**Вопрос:** как лучше выбрать B?)

При добавлении элементов важно поддерживать указанную выше структуру. Вставку узла будем выполнять по индексу, на котором должен стоять новый элемент. При этом сначала нужно решить, будет ли новый узел большим — это можно делать случайным образом, так чтобы поддерживать желаемую долю больших узлов. Затем нужно найти по индексу последний больший и малый узлы перед тем местом, куда нужно вставить новый узел. Наконец, можно вставить новый узел, корректно обновив все ссылки и расстояния так, чтобы сохранилась структура списка.

Предлагается реализовать эту структуру данных, и обработать с её помощью серию из N запросов двух типов (в описаниях L — текущее количество узлов в списке):

0. Вставить на k-ую позицию новый узел с заданным значением V. Гарантируется, что во всех запросах k лежит в пределах от 0 до L.
1. Вывести значение узла, находящегося на k-ой позиции в списке. Гарантируется, что во всех запросах k лежит в пределах от 0 до $L - 1$.

Решение в целом должно работать за время $O(N\sqrt{N})$.

Формат входных данных

В первой строке задано целое число N — общее количество запросов ($1 \leqslant N \leqslant 10^5$). В каждой из следующих N строк дан один запрос. Запрос вставки задаётся тремя целыми числами 0 k V, а запрос на вывод элемента задаётся двумя целыми числами 1 k. Все числа V в списке целые, по модулю не превышают 10^9.

Формат выходных данных

Для каждого запроса на вывод (типа 1) нужно вывести значение k-ого узла в списке.

Пример

input.txt	output.txt
10 0 0 7 0 0 5 0 1 3 0 0 2 1 0 1 1 1 2 1 3 0 1 -5 1 2	2 5 3 7 5

Комментарий

Несмотря на формально лучшую асимптотику, полученная структура данных будет обрабатывать заданные запросы не быстрее простого массива.

Задача 10. Список с индексами++

Источник:	повышенной сложности*
Имя входного файла:	`input.txt`
Имя выходного файла:	`output.txt`
Ограничение по времени:	5 секунд*
Ограничение по памяти:	разумное

Данная задача посвящена дальнейшему улучшению структуры данных из задачи "Список с индексами".

Вместо деления узлов лишь на "большие" и "маленькие", припишем каждому узлу "высоту". Высота определяет, сколько в узле хранится ссылок на следующие элементы. В предыдущей задаче список имел только два уровня: "маленькие" узлы имели высоту 1, а "большие" — высоту 2. В этой задаче предлагается завести 20 уровней (на самом деле примерно $\log_2 N$), распределяя высоты от 1 до 20.

В узле высоты h нужно хранить массив из h указателей вперёд вместе с расстояниями. k-ый указатель смотрит на следующий в списке элемент, высота которого хотя бы k (здесь полагаем, что k считается с единицы). Для идеальных результатов узлов высоты k должно быть примерно 2^{-k} по доле. Этого можно достичь, если при добавлении каждого нового узла бросать монетку до тех пор, пока не выпадёт орёл, а высоту устанавливать как количество выполненных бросков.

При правильной реализации этой структуры можно выполнять поиск по индексу и добавление узлов примерно за $O(\log N)$. Такая структура называется "список с пропусками" / "skip lists": ссылка.

В данной задаче нужно обработать N таких же запросов, как в задаче "Список с индексами", но теперь за время $O(N \log N)$.

Формат входных данных

В первой строке задано целое число N — общее количество запросов ($1 \leqslant N \leqslant 10^6$). В каждой из следующих N строк дан один запрос. Запрос вставки задаётся тремя целыми числами $0\ k\ V$, а запрос на вывод элемента задаётся двумя целыми числами $1\ k$. Все числа V в списке целые, по модулю не превышают 10^9.

Формат выходных данных

Для каждого запроса на вывод (типа 1) нужно вывести значение k-ого узла в списке.

Пример

input.txt	output.txt
10 0 0 7 0 0 5 0 1 3 0 0 2 1 0 1 1 1 2 1 3 0 1 -5 1 2	2 5 3 7 5