Внимание! Некоторые задачи в этом наборе тестируются с помощью специальной системы — emailtester. Это позволяет более тщательно проверять, как ваши решения управляют памятью. Чтобы решение было засчитано, требуется, чтобы вся выделенная в течение работы программы память была освобождена.

Исторически emailtester – это система проверки задач по электронной почте, никак не связанная с привычной nsuts. Сейчас emailtester интегрирован с nsuts, так что решения отправляются через уже привычный вам интерфейс, только при отправке необходимо выбрать `"E-mail Tester"` в списке компиляторов.

После отправки ваше решение будет обработано системой примерно через одну-две минуты. В результате будет выставлен один из следующих вердиктов:

- `ACCEPTED` – решение успешно принято, задача сдана;
- `Wrong Answer (#1)` – решение не принято. Конкретную причину, по которой оно не было принято, можно узнать, внимательно изучив текстовый лог проверки вашего решения. Лог проверки можно найти на вкладке `"Результаты"`, кликнув на ссылку в столбце `"Решения"`;
- `Presentation Error (#1)` – решение неверно сформировано (скорее всего отправлен файл неправильного формата).

Если никакой ответ не приходит в течение 10-15 минут, тогда скорее всего на проверяющей машине что-то упало – такое бывает. Срочно напишите об этом лектору, желательно указав идентификатор посылки и название задачи.

Задача 1. Распечатать список

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

Дан односвязный список, который хранится в массиве. Массив состоит из узлов, в каждом узле лежит какое-то вещественное значение и индекс следующего элемента списка в том же массиве. Нужно распечатать вещественные значения всех узлов в порядке их следования в связном списке.

Формат входных данных

В первой строке содержится два целых числа: N — количество узлов в массиве и F — индекс первого элемента связного списка ($0 \leqslant F < N \leqslant 100\,000$).

В каждой k-ой из оставшихся N строк записано содержимое k-ого элемента массива. Сначала записывается значение (вещественное число с трёмя или меньше знаками после точки, по модулю не превышает 10^4), а затем индекс следующего элемента.

Все индексы элементов нумеруются с нуля. В последнем узле связного списка индекс следующего элемента равен -1.

Формат выходных данных

Нужно вывести N строк, по одному вещественному числу в каждой строке — значения всех узлов в порядке их следования в связном списке. Вещественные числа нужно выводить хотя бы с тремя знаками после точки, например используя формат `"%0.3lf"`.

Пример

input.txt	output.txt
5 3 4.283 2 2.718 4 5.0 -1 1.111 1 3.141 0	1.111 2.718 3.141 4.283 5.000

Пояснение к примеру

В примере порядок элементов получается: 3 (1.111) -> 1 (2.718) -> 4 (3.141) -> 0 (4.283) -> 2 (5.0).

Задача 2. Односвязный список

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

Требуется реализовать односвязный список на массиве. В каждом узле списка хранится строковое значение и индекс следующего узла. 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:

0. *Добавление узла.* При этом указывается индекс узла, после которого нужно вставить новый узел, и строковое 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. *Удаление узла.* При этом указывается индекс узла, и удалить нужно тот узел, который идёт сразу после него. Гарантируется, что узел с указанным индексом не последний. Если указан индекс -1, значит нужно удалить самый первый элемент списка (гарантируется, что он есть). После выполнения операции нужно вывести строковое значение удалённого узла.

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

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

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

Наконец, в тесте идут Q строк, которые описывают операции над списком. В каждой строке сначала записан тип операции: 0 — добавление, 1 — удаление. Затем указан индекс узла, после которого нужно вставить/удалить узел. Если описывается операция вставки узла, то в конце также задано строковое значение нового узла.

У каждого узла строковое значение имеет длину от 1 до 7 символов включительно, и состоит из произвольных печатаемых символов ASCII кроме пробела (такие символы имеют код от 33 до 126 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. строковые значения всех узлов списка после выполнения операций (в порядке их следования в списке)
4. снова строка "==="

Пример

input.txt	output.txt
3 5 3 0 4.283 2 2.718 4 5.0 -1 1.111 1 3.141 0 1 0 5 zero -1 0 -1 one 0 -1 two 0 1 three 0 0 four 1 2 4 2 2 45 1 %drill# 3 &qw6: 0 *a-+r -1 1 1 0 2 \num\	=== 1.111 2.718 3.141 4.283 5.0 === 1 2 3 4 one === two three zero four === *a-+r 4 === &qw6: \num\ 45 %drill# ===

Пояснение к примеру

В примере три теста.

Первый тест полностью совпадает с примером к задаче “Распечатать список”: дано 5 узлов и 0 операций. Т.к. операций нет, в ответе записана сразу строка "===". Потом записан ответ как в задаче “Распечатать список” и ещё одна строка "===".

Во втором тесте изначально есть только один узел. Заметим, что в каждом узле значение равно индексу, записанному по-английски: `zero`, `one`, `two`, `three`, `four`.

Первые четыре операции задают вставку элементов: сначала вставляется два узла в начало, получается список `two`, `one`, `zero`. Затем вставляется узел после узла `one` и ещё один узел после узла `zero`, получается список `two`, `one`, `three`, `zero`, `four`. Последняя операция удаления: удаляется узел, который стоит **после** узла `two`, то есть узел `one`.

Задача 3. Хранение строк

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

Внимание: эта задача проверяется на **emailtester**.

Нужно реализовать систему хранения строк в динамической памяти.

Система должна обрабатывать три вида запросов/операций:

0. *Создать строку*: указывается длина создаваемой строки и собственно содержимое строки. Нужно выделить блок памяти при помощи `malloc` и записать в него содержимое.
1. *Удалить строку*: указывается идентификатор ранее созданной строки. Нужно освободить память, выделенную ранее для этой строки, при помощи `free`.
2. *Вывести строку*: указывается идентификатор ранее созданной строки. Нужно распечатать содержимое этой строки в выходной файл.
3. *Сколько символов в строке*: указывается идентификатор ранее созданной строки и один символ. Нужно вывести в выходной файл одно целое число: сколько раз этот символ встречается в указанной строке.

Идентификатором строки является номер запроса на её создание в общей нумерации запросов. При обращении к строке по идентификатору гарантируется, что эта строка ещё **не** была удалена.

Замечание: Не все созданные строки удаляются в запросах. В целях аккуратности удалите все остающиеся строки при помощи `free` в конце программы.

Формат входных данных

В первой строке записано целое число N — количество количество запросов ($1 \leqslant N \leqslant 10^5$). В остальных N строках описаны запросы, по одному запросу в строке.

Запрос начинается с целого числа t, обозначающего тип запроса. Если $t = 0$, то это запрос создания, и тогда далее указана длина строки l и сама строка ($1 \leqslant l \leqslant 10^5$). Если $t = 1$, то это запрос удаления, а если $t = 2$ — то это запрос на вывод. В обоих случаях далее указано целое число k — идентификатор строки ($0 \leqslant k < N$). Если $t = 3$, то это запрос о количестве символов, и тогда далее указан идентификатор строки k и ещё один символ через пробел.

Все строки состоят из произвольных печатаемых символов ASCII кроме пробела (коды от 33 до 126 включительно). То же верно и про символы в запросах на количество.

Сумма всех длин l по всем запросам создания не превышает $5 \cdot 10^5$. Суммарный размер всех строк, которые вам нужно вывести, не превышает $5 \cdot 10^5$. Сумма длин строк по всем запросам о количестве символов не превышает 10^8.

Пример

input.txt	output.txt
12 0 3 aba 2 0 3 0 a 3 0 b 1 0 0 6 malloc 0 5 %d%s% 2 1 2 2 1 1 3 2 % 2 2	aba 2 1 malloc %d%s% 3 %d%s%

Пояснение к примеру

Сначала создаётся строка “aba” с идентификатором 0. Далее она распечатывается, и выводится количество букв ’a’ и ’b’ в ней. Наконец, нулевая строка удаляется.

Затем создаются ещё две строки: строка “malloc” и строка “%d%s%” с идентификаторами 1 и 2 соответственно. Потом они распечатываются и строка “malloc” удаляется. В конце выводится количество процентов в строке “%d%s%” и она распечатывается ещё раз.

Задача 4. Двусвязный список

Источник:	основная
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 5. 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 6. Разделить на слова

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

Внимание: эта задача проверяется на **emailtester**.

Требуется реализовать функции для выделения слов из заданной строки. Слова отделены друг от друга символами-разделителями, которые передаются в строку как параметр.

Сигнатура функций должна быть такой:

```
typedef struct Tokens_s {
    int num;         //количество слов в строке
    char **arr;      //массив слов (каждый элемент -- строка, т.е. char*)
} Tokens;
//tokens: структура, в которую нужно положить результаты
//str: строка, в которой нужно искать слова
//delims: все символы-разделители в виде строки
void tokensSplit(Tokens *tokens, const char *str, const char *delims);
//удаляет все ресурсы внутри tokens
void tokensFree(Tokens *tokens);
```

Память для массива слов `tokens->arr` следует выделять динамически ровно на столько элементов, сколько слов в строке. Поскольку заранее это количество неизвестно, то нужно запускать алгоритм два раза: первый раз, чтобы узнать сколько слов, и второй раз, чтобы записать слова в массив.

Ваша реализация вышеописанных функцию должна работать согласно следующим договорённостям:

1. Вызывающий гарантирует, что параметр `tokens` **не** нулевой (для обеих функций) и указывает на структуру `Tokens`.
2. Если при вызове `tokensSplit` указатель `tokens->arr` нулевой, то внутри функции нужно только посчитать количестов слов и записать его в `tokens->num`.
3. Если при вызове `tokensSplit` указатель `tokens->arr` **не** нулевой, то он должен указывать на массив, в который точно войдут все слова. В этом случае реализация функции должна записать в `tokens->num` количество слов, а в `tokens->arr[i]` записать i-ое слово, самостоятельно выделив под него память с помощью `malloc`.
4. Функция `tokensFree` должна удалять массив слов и сами строки-слова с помощью `free`. При этом программа должна работать корректно, даже если эту функцию случайно вызовут два или три раза подряд.

Таким образом, вызывающий может завести структуру `tokens`, потом определить количество слов первым вызовом `tokensSplit`, затем выделить память на массив `tokens->arr`, и, наконец, найти все слова вторым вызовом `tokensSplit`.

С помощью этих функций нужно решить тестовую задачу. Дана одна строка длиной до 10^6, состоящая из букв латинского алфавита и знаков препинания четырёх типов: точка, запятая, точка с запятой, двоеточие. Нужно найти слова в этой строке, состоящие из букв, и вывести их в файл в таком же формате, как показано в примере.

Пример

input.txt	output.txt
..ko,.Privet:kreved,.,;ko:;,.	4 ko Privet kreved ko

Комментарий

Следует выводить слова ровно в том порядке, в котором они встречаются в строке.

Задача 7. Список с указателями

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

Внимание: эта задача проверяется на **emailtester**.

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.

Узлам присваиваются индексы в порядке их создания. Самый первый созданный узел имеет индекс 0, следующая операция создания имеет индекс 1, и так далее. Индекс узла никогда не переиспользуется, даже после того, как узел удаляют из списка.

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 8. 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

Внимание: эта задача проверяется на **emailtester**.

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 9. Список с индексами++

Источник:	повышенной сложности*
Имя входного файла:	`input.txt`
Имя выходного файла:	`output.txt`
Ограничение по времени:	3 секунды*
Ограничение по памяти:	разумное

Внимание: эта задача проверяется на **emailtester**.

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