Авторы:

Медведев Михаил Геннадиевич – кандидат физико-математических наук, доцент кафедры математической информатики факультета кибернетики Киевского национального университета имени Тараса Шевченко.

Присяжнюк Анатолий Васильевич – учитель-методист высшей категории специализированной школы с углубленным изучением информатики № 17 г. Бердичев Житомирской области.

Жуковский Сергей Станиславович – старший преподаватель кафедры прикладной математики и информатики Житомирского государственного университета имени Ивана Франко, учитель информатики городского лицея №25 имени Н. А. Щорса.

Задачи:

Стек: 693,1776,1871,1872,2479
Очередь: 694
Множество и мультимножество: 555,790,1225,1226,1227,1228,2661, 3004
Отображение: 1211,1868,2040
Другое: 291

ОГЛАВЛЕНИЕ

УСЛОВИЯ ЗАДАЧ

291. Парковка

Парковка имеет n мест, пронумерованных от 1 до n включительно. Парковка открывается пустой каждое утро и работает на протяжении дня следующим образом. Когда автомобиль приезжает на парковку, парковщик проверяет, есть ли свободные места. Если таковых нет, автомобиль ожидает возле въезда до тех пор, пока освободится какое-то место. Если есть свободное место, или как только оно освобождается, автомобиль занимает свободное парковочное место. Если есть несколько свободных парковочных мест, автомобиль занимает место с наименьшим номером. Когда приезжают парковаться другие автомобили, но уже есть ожидающий автомобиль, они выстраиваются в очередь на въезде в том порядке, в котором приехали. После того, как освобождается парковочное место, его занимает первый автомобиль из очереди (то есть тот, который прибыл парковаться первым).

Стоимость парковки одного автомобиля в долларах определяется как произведение веса этого автомобиля в килограммах на тариф его парковочного места. Стоимость парковки автомобиля не зависит от того, сколько времени этот автомобиль находится на парковке.

Парковщик знает, что сегодня на парковку приедет m автомобилей, и он знает порядок их приезда и отъезда. Помогите ему подсчитать, сколько долларов он сегодня заработает.

Напишите программу, которая по заданным тарифам парковочных мест, весам автомобилей и порядку, в котором автомобили приезжают и уезжают, определяет доход парковки в долларах.

Вход. Первая строка содержит два целых числа – количество парковочных мест n ($1 \le n \le 100$) и количество автомобилей m ($1 \le m \le 2{,}000$), разделенные пробелом.

Следующие n строк описывают тарифы парковочных мест, s-ая из этих строк содержит одно целое число r_s ($1 \le r_s \le 100$) – тариф парковочного места с номером s в долларах за килограмм.

Следующие m строк описывают веса автомобилей. Автомобили пронумерованы в произвольном порядке от 1 до m включительно, k-ая из этих m строк содержит одно целое число w_k ($1 \le w_k \le 10{,}000$) – вес автомобиля с номером k в килограммах.

Следующие $2m$ строк описывают приезд и выезд всех автомобилей в хронологическом порядке. Положительное целое число i показывает, что автомобиль с номером i приезжает на парковку. Отрицательное целое число $-i$ показывает, что автомобиль с номером i уезжает с парковки. Никакой автомобиль не выезжает с парковки до своего приезда, и все автомобили от 1 до m включительно появятся в этой последовательности строк ровно 2 раза, один раз как приезжающий, и второй – как выезжающий. К тому же, никакой из автомобилей не выедет с парковки, пока не займет место на парковке (то есть, никакой автомобиль не уедет пока стоит в очереди).

Выход. Одно целое число – общее количество долларов, которое заработает сегодня парковщик.

Пример входа

```
3 4
2
3
5
200
100
300
800
```

```
3
2
-3
1
4
-4
-2
-1
```

Пример выхода
```
5300
```

555. Ближайшие точки

Антон в школе начал изучать математику. Его внимание привлекло новое для него понятие числовой прямой. Антон быстро научился вычислять расстояния между двумя точками на этой прямой, задавать отрезки и интервалы на ней.

Готовясь к контрольной работе, Антон столкнулся со следующей задачей: На числовой прямой задано n точек. Необходимо найти среди них две ближайшие. Расстояние между двумя точками и числовой прямой x и y равно $|x-y|$.

Требуется написать программу, которая поможет Антону решить поставленную задачу.

Вход. Первая строка содержит количество точек n ($2 \le n \le 10^5$). Вторая строка содержит n различных целых чисел x_i – координаты заданных точек числовой прямой. Числа в строке разделены пробелом. Значение любой координаты x_i не превосходит 10^9 по абсолютной величине.

Выход. В первой строке следует вывести минимальное расстояние между двумя заданными точками. Во второй строке необходимо вывести номера точек, которым соответствует найденное расстояние. Точки нумеруются натуральными числами от 1 до n в том же порядке, в котором они заданы на входе. Если ответов несколько, выведите любой из них.

Пример входа
```
5
10 3 6 2 5
```

Пример выхода
```
1
2 4
```

693. Минимум в стеке

На вход программы подается набор операций со стеком. Каждая операция состоит в добавлении или удалении элемента из стека. После выполнения каждой операции найдите наименьшее число, которое находится в стеке. Сложите все полученные числа и получите ответ. Если после некоторой операции стек оказался пуст, то ничего не прибавляйте к ответу. Если выполнить удаление невозможно, так как стек пуст, то не выполняйте его.

Вход. Входные данные генерируются в самой программе. На вход подаются параметры для генерации входной последовательности.

Первое число содержит количество операций n ($1 \le n \le 10^6$) со стеком. Затем следуют четыре неотрицательных целых числа a, b, c, x_0, не превосходящие 10000.

Для получения входных данных сгенерируем последовательность x.

Первое число в генерируемой последовательности x_1. Каждое следующее число вычисляется из предыдущего по формуле:

$$x_i = (a \cdot x_{i-1}^2 + b \cdot x_{i-1} + c) / 100 \bmod 10^6,$$

где '/' – операция целочисленного деления, а 'mod' – остаток от деления.

Если $x_i \bmod 5 < 2$, то необходимо удалить число из стека. В противном случае нужно добавить в стек число x_i.

Выход. Выведите результирующую сумму.

Пример входа
```
2 0 0 1 81
```

Пример выхода
```
0
```

694. Минимум в очереди

На вход вашей программе подается набор операций с очередью. Каждая операция - это добавить или удалить элемент из очереди. После выполнения каждой операции вычислите наименьшее из всех чисел. Сложите все полученные числа и получите ответ. Если после некоторой операции очередь оказалась пуста, то ничего не прибавляйте к ответу. Если выполнить удаление невозможно из-за того что очередь пуста, то не выполняйте его.

Вход. В этой задаче входные данные будут генерироваться прямо в вашей программе. На вход программе будут поданы параметры, чтобы вы смогли сгенерировать входную последовательность. Первое число во входном файле n ($1 \le n \le 10^6$) – количество операций проводимых с очередью. Затем идут четыре неотрицательных числа a, b, c, x_0 не превосходящие 10000.

Для получения входных данных сгенерируем последовательность x.

Первое число в генерируемой последовательности – x_1. Первое как и каждое очередное число вычисляется из предыдущего:

$$x_i = (a \cdot x_{i-1}^2 + b \cdot x_{i-1} + c) / 100 \bmod 10^6,$$

где '/' – операция целочисленного деления, а 'mod' - остаток от деления.

Если $x_i \bmod 5 < 2$, то необходимо удалить число из очереди, в противном случае нужно добавить в очередь число x_i.

Выход. Выведите единственное число – искомую сумму.

Пример входа
```
2 0 0 1 81
```

Пример выхода
```
0
```

790. Двойная очередь

Недавно образованная Балканская Инвестиционная Банковская Группа (БИГ-Банк) открыла новый офис в Бухаресте, оснащенный современной вычислительной техникой, предоставленной компанией IBM Румыния, с использованием современных информационных технологий. Как правило, каждый клиент банка идентифицируется натуральным числом k. По приходу в банк за получением услуг, он или она получает некоторое натуральное число – приоритет p. Одно из изобретений молодых менеджеров банка шокировало инженера программного обеспечения по обслуживанию системы. Они решили сломать традицию, предложив обслуживать первым клиента не только с наибольшим приоритетом, но и с наименьшим. Известно, что система на вход получает следующие типы запросов:

0 Система обслуживания клиентов останавливается
1 *k p* Добавить клиента *k* с приоритетом *p* в список ожидания
2 Обслужить клиента с наибольшим приоритетом и удалить из списка ожидания
3 Обслужить клиента с наименьшим приоритетом и удалить из списка ожидания

Вам следует помочь инженеру программного обеспечения банка, написав программу обслуживания клиентов согласно указанному принципу.

Вход. Каждая входная строка содержит один из возможных запросов; только последняя строка содержит требование остановить работу системы (код 0). Если приходит запрос на включение клиента в очередь (код 1), то считайте, что других запросов по этому клиенту, или по клиенту с таким же приоритетом на данный момент не существует. Значение k всегда меньше 10^6, а приоритет p меньше 10^7. Клиент может приходить и обслуживаться несколько раз, каждый раз получая разное значение приоритета.

Выход. Для каждого запроса с кодом 2 или 3 программа должна вывести в отдельной строке идентификатор обслуженного клиента. Если поступает запрос при пустой очереди на обслуживание, то следует вывести ноль (0).

Пример входа
```
2
1 20 14
1 30 3
2
1 10 99
3
2
2
0
```

Пример выхода
```
0
20
30
10
0
```

1211. Бесконечная последовательность

Определим бесконечную последовательность A следующим образом:

$$A_0 = 1,$$
$$A_i = A_{[i/p]} + A_{[i/q]}, \; i \geq 1$$

По заданным n, p, q необходимо вычислить A_n.

Вход. Три целых числа n, p, q ($0 \leq n \leq 10^{12}$, $2 \leq p, q \leq 10^9$).

Выход. Значение A_n.

Пример входа
```
10000000 3 3
```

Пример выхода
```
32768
```


1225. Черный ящик

Черный Ящик представляет собой примитивную базу данных. Он может хранить массив целых чисел, а также имеет специальную переменную i. В начальный момент Черный Ящик пустой, переменная i равна 0. Черный Ящик обрабатывает последовательность команд (транзакций). Существует два типа транзакций:

ADD(x): положить элемент x в Черный Ящик;

GET: увеличить i на 1 и вывести i-ый минимальный элемент среди всех чисел, находящихся в Черном Ящике;

Помните, что i-ый минимальный элемент находится на i-ом месте после того как все элементы черного ящика будут отсортированы в неубывающем порядке.

Рассмотрим работу Черного Ящика на примере:

шаг	операция	i	содержимое черного ящика	результат
1	**ADD**(3)	0	3	
2	**GET**	1	3	3
3	**ADD**(1)	1	1, 3	
4	**GET**	2	1, 3	3
5	**ADD**(-4)	2	-4, 1, 3	
6	**ADD**(2)	2	-4, 1, 2, 3	
7	**ADD**(8)	2	-4, 1, 2, 3, 8	
8	**ADD**(-1000)	2	-1000, -4, 1, 2, 3, 8	
9	**GET**	3	-1000, -4, 1, 2, 3, 8	1
10	**GET**	4	-1000, -4, 1, 2, 3, 8	2
11	**ADD**(2)	4	-1000, -4, 1, 2, 2, 3, 8	

Необходимо разработать эффективный алгоритм выполнения заданной последовательности транзакций. Максимальное количество транзакций **ADD** и **GET** равно 30000 (каждого типа).

Опишем последовательность транзакций двумя целочисленными массивами:

1. A(1), A(2), ..., A(m): последовательность элементов, которая будет добавляться в Черный Ящик. Элементами являются целые числа, по модулю не большие 2 000 000 000, $m \leq 30000$. Для выше описанного примера A = (3, 1, -4, 2, 8, -1000, 2).
2. u(1), u(2), ..., u(n): последовательность указывает на количество элементов в Черном Ящике в момент выполнения первой, второй, ... n-ой транзакции GET. Для выше описанного примера u = (1, 2, 6, 6).

Работа Черного Ящика предполагает, что числа в последовательности u(1), u(2), ..., u(n) отсортированы в неубывающем порядке, $n \leq m$, а для каждого p ($1 \leq p \leq n$) имеет место неравенство $p \leq u(p) \leq m$. Это следует из того, что для p-го элемента последовательности u мы выполняем GET транзакцию, которая выводит p-ый минимальный элемент из набора чисел A(1), A(2), ..., A(u(p)).

Вход. Состоит из следующего набора чисел: m, n, A(1), A(2), ..., A(m), u(1), u(2), ..., u(n). Все числа разделены пробелами и (или) символом перевода на новую строку.

Выход. Вывести ответы Черного Ящика на последовательность выполненных транзакций. Каждое число должно выводиться в отдельной строке.

Пример входа

```
7 4
3 1 -4 2 8 -1000 2
1 2 6 6
```

Пример выхода

```
3
3
1
2
```

1226. Обмен иностранцами

Ваша неприбыльная организация координирует программу по обмену студентами. И ей нужна Ваша помощь.

Программа обмена работает следующим образом. Каждый из участников дает информацию о месте своем проживания и месте, куда бы он хотел переехать. Программа считается успешной, если каждый студент найдет для обмена подходящего партнера. Другими словами, если некоторый студент желает переехать из A в B, то обязательно должен быть другой студент, который хочет переехать из B в A. Это простая задача, если участников программы всего 10. Но что делать, если их будет 100001?

Вход. Первая строка содержит количество тестов t. Первая строка каждого теста содержит количество студентов n ($1 \le n \le 100001$), за которыми следуют n строк, описывающие данные по обмену. Каждая из этих строк содержит информацию об одном студенте – два целых числа, разделенные пробелом, соответствующих текущему месту проживания студента и месту, куда он желает переехать. Места описываются неотрицательными целыми числами, не большими 10^9. Ни у одного из кандидатов место проживания и место желаемого переезда не совпадают.

Выход. Для каждого теста в отдельной строке вывести "YES" если существует возможность успешно выполнить программу обмена и "NO" иначе.

Пример входа

```
2
10
1 2
2 1
3 4
4 3
100 200
200 100
57 2
2 57
1 2
2 1
10
1 2
3 4
5 6
7 8
9 10
11 12
13 14
15 16
17 18
19 20
```

Пример выхода

```
YES
NO
```


1227. Первый словарь Энди

У Энди есть мечта – он хочет создать свой собственный словарь. Но для него это не простая задача, так как количество известных ему слов достаточно мало. Вместо того чтобы вспоминать слова, у Энди возникла великолепная идея. Он решил взять с полки свою любимую книгу и выписать из нее все различные слова. Далее он расположил все слова в алфавитном порядке. Конечно, такая работа занимает много времени, поэтому ему может помочь компьютерная программа.

Вам необходимо написать программу, которая выводит все различные слова в тексте. Словом называется последовательность заглавных и прописных букв латинского алфавита. Слово может состоять из одной буквы. Более того, программа не должна быть чувствительной к регистру. Например, слова "Apple", "apple" или "APPLE" считаются одинаковыми.

Вход. На вход подается текст, который содержит не более 5000 строк. Каждая строка содержит не более 200 символов.

Выход. Вывести следует список разных слов, встречающихся в тексте, каждое слово выводить в отдельной строке. Слова должны содержать только прописные буквы и быть отсортированными в алфавитном порядке. Количество различных слов во входном тексте не превышает 6000.

Пример входа

```
Adventures in Disneyland

Two blondes were going to Disneyland when they came to a fork in the
road. The sign read: "Disneyland Left."

So they went home.
```

Пример выхода

```
a
adventures
blondes
came
disneyland
fork
going
home
in
left
read
road
sign
so
the
they
to
two
went
were
when
```

1228. Сложить все

Стоимость сложения двух чисел равна их сумме. То есть для того чтобы к 1 прибавить 10, следует заплатить 11. В задаче требуется сложить все заданные числа, потратив при этом наименьшее количество денег.

Например, складывать числа 1, 2 и 3 можно одним из трех способов:

1 + 2 = 3, стоимость 3	1 + 3 = 4, стоимость 4	2 + 3 = 5, стоимость 5
3 + 3 = 6, стоимость 6	4 + 2 = 6, стоимость 6	1 + 5 = 6, стоимость 6
Общая стоимость 9	Общая стоимость 10	Общая стоимость 11

Первый способ сложения самый дешевый.

Вход. Первая строка каждого теста содержит количество складываемых чисел n ($2 \le n \le 500$). Вторая строка теста содержит n чисел, каждое из которых не более 100000. Последняя строка содержит $n = 0$ и не обрабатывается.

Выход. Для каждого теста найти наименьшую стоимость, за которую можно сложить все заданные n чисел.

Пример входа

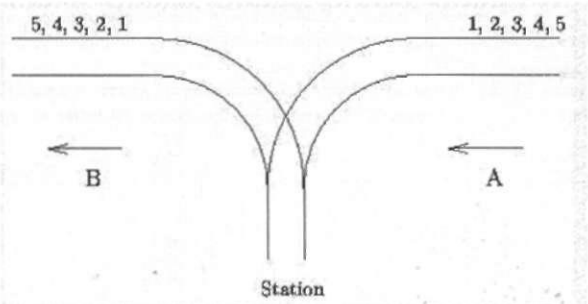
```
3
1 2 3
4
1 2 3 4
0
```

Пример выхода

```
9
19
```

1776. Рельсы

В городе PopPush находится известная железнодорожная станция. Страна, в которой находится город, невероятно холмистая. Станция была построена в прошлом веке. К сожалению, в то время средства для постройки были крайне ограничены, поэтому удалось построить только одну железнодорожную колею. Более того, выяснилось, что станция может быть только тупиковой (см. рисунок), и из-за отсутствия свободного места может иметь только одну колею.



Местная традиция гласит, что каждый поезд, приходящий со стороны A, продолжает свое движение в направлении B, при этом его вагоны перестанавливаются в некотором порядке. Предположим, что каждый поезд, приходящий из направления A, имеет $n \le 1000$ вагонов, пронумерованных в возрастающем порядке 1, 2, ..., n. Ответственный за реорганизацию вагонов должен знать, возможно ли перевезти их в направлении B в порядке $a_1, a_2, ..., a_n$. Помогите ему написать программу, которая определит, возможна ли такая перестановка вагонов. Вагоны можно отсоединять от поезда до того как они попадут на станцию и можно их отдельно передвигать пока все они не буду находиться в направлении B. На станции в любой момент времени может находиться любое количество вагонов. Но если вагон зашел на станцию, он уже не может вернуться на колею в направлении A, а также если он уже выехал в направлении B, то уже не может вернуться на станцию.

Вход. Состоит из нескольких тестов. Каждый тест кроме последнего описывает один поезд и возможно несколько требований для его реорганизации. Первая строка теста содержит целое число n. Каждая из следующих строк теста содержит перестановку 1, 2, ..., n. Последняя строка блока содержит 0.

Последний тест состоит из единственной строки, содержащей 0.

Выход. Для каждой входной строки, содержащей перестановку чисел, следует вывести Yes, если можно совершить указанную перестановку вагонов, и No иначе. После вывода ответов на все перестановки каждого теста следует вывести пустую строку. Для последнего нулевого теста ничего выводить не следует.

Пример входа

```
5
1 2 3 4 5
5 4 1 2 3
0
6
6 5 4 3 2 1
0
0
```

Пример выхода

```
Yes
No

Yes
```

1868. Функция

Вычислите функцию:

$$f(n) = \begin{cases} 1 & \text{если } n \le 2 \\ f(\lfloor 6 * n/7 \rfloor) + f(\lfloor 2 * n/3 \rfloor) & \text{если } n \bmod 2 = 1 \\ f(n-1) + f(n-3) & \text{если } n \bmod 2 = 0 \end{cases}$$

Вход. Одно натуральное число n ($1 \le n \le 10^{12}$).

Выход. Значение f(n), взятое по модулю 2^{32}.

Пример входа

```
7
```

Пример выхода

```
10
```

1871. Белка и бамбук

Белка решила отправиться в кругосветное путешествие. Попав в тропики, она обнаружила, что жёлуди находить стало труднее. Зато она нашла отличный стебель бамбука, и теперь вместо того, чтобы каждый день таскать жёлуди по одному от одного дупла до другого, носит их с собой в бамбуке.

Бамбук – это трубка, один конец которой закрыт, а с другого конца можно класть или вынимать жёлуди. Диаметр трубки достаточно мал, поэтому если положить в неё жёлуди в определённом порядке, вынимать их можно только в обратном порядке.

Когда белка находит жёлудь, она сразу кладёт его в бамбук. Кроме того, время от времени голод заставляет белку достать один жёлудь из бамбука и съесть его; из-за устройства бамбука это будет тот из желудей в нём, который белка нашла позже всего.

Белка очень любит копить жёлуди в бамбуке. Поэтому каждый раз, когда приходится доставать из бамбука очередной жёлудь, она испытывает печаль. Однако мы знаем, как можно её утешить! Жёлуди характеризуются для белки качеством – целым числом от 1 до 10^6. Когда белка достала очередной жёлудь, ей будет приятно знать, каково максимальное значение качества для всех желудей, которые в бамбуке ещё остались. Ваша задача состоит в том, чтобы снабдить её такой информацией.

Вход. В первой строке ввода содержится одно целое число n – количество событий ($1 \le n \le 100000$). Каждая из следующих n строк содержит по числу, описывающему событие. Если число положительное, то оно означает, что белка нашла жёлудь и положила его в свой бамбук. Если же число равно нулю, то оно означает, что белка проголодалась и вынула один жёлудь из бамбука. Числа во входном файле целые и не превосходят 10^6. Гарантируется, что после первого же запроса бамбук никогда не пуст.

Выход. Для каждого доставания жёлудя из бамбука выведите строку, содержащую одно целое число – максимальное значение качества для всех желудей, оставшихся в этот момент в бамбуке.

Пример входа

```
8
3
2
4
0
4
3
0
0
```

Пример выхода

```
3
4
3
```

1872. Снеговики

Зима. 2012 год. На фоне грядущего Апокалипсиса и конца света незамеченной прошла новость об очередном прорыве в областях клонирования и снеговиков: клонирования снеговиков. Вы конечно знаете, но мы вам напомним, что снеговик состоит из нуля или более вертикально поставленных друг на друга шаров, а клонирование — это процесс создания идентичной копии (клона).

В городе Местячково учитель Андрей Сергеевич Учитель купил через интернет-магазин "Интернет-магазин аппаратов клонирования" аппарат для клонирования снеговиков. Теперь дети могут играть и даже играют во дворе в следующую игру. Время от времени один из них выбирает понравившегося снеговика, клонирует его и:

- либо добавляет ему сверху один шар;
- либо удаляет из него верхний шар (если снеговик не пустой).

Учитель Андрей Сергеевич Учитель записал последовательность действий и теперь хочет узнать суммарную массу всех построенных снеговиков.

Вход. Первая строка содержит количество действий n ($1 \le n \le 200000$). В строке номер $i + 1$ содержится описание действия:

t m – клонировать снеговика номер t ($0 \le t < i$) и добавить сверху шар массой m ($0 < m \le 1000$);

t 0 — клонировать снеговика номер t ($0 \le t < i$) и удалить верхний шар. Гарантируется, что снеговик не пустой.

В результате действия i, описанного в строке $i + 1$ создастся снеговик номер i. Изначально имеется пустой снеговик с номером ноль.

Все числа во входном файле целые.

Выход. Выведите суммарную массу построенных снеговиков.

Пример входа

```
8
0 1
1 5
2 4
3 2
4 3
5 0
6 6
1 0
```

Пример выхода

```
74
```

2040. Обычная перестановка

По заданным двум строкам a и b следует вывести такую строку x наибольшей длины, которая одновременно является подстрокой перестановки a и подстрокой перестановки b.

Вход. Состоит из нескольких тестов, каждый их которых содержит две строки. То есть строки 1 и 2 – это первый тест, строки 3 и 4 – второй тест и т.д. Каждая строка состоит из символов нижнего регистра, причём первой строкой в паре является a, а второй строкой b. Максимальная длина каждой строки 1000 символов.

Выход. Для каждого теста следует в отдельной строке вывести строку x. Если таких строк несколько, то вывести наименьшую в алфавитном порядке.

Пример входа

```
pretty
women
walking
down
the
streeт
```

Пример выхода

```
e
nw
et
```

2479. Баланс скобок

Имеется строка, содержащая скобки () и []. Скобочное выражение считается правильным, если:

- оно является пустым
- если A и B правильны, то AB правильно
- если A правильно, то (A) и [A] правильны

Напишите программу, которая по входной строке, содержащей скобочное выражение, определит корректно ли оно. Длина строки не больше 128 символов.

Вход. Первая строка содержит количество тестов n. Каждая из следующих n строк содержит выражение, состоящее из скобок () и [].

Выход. Для каждого теста вывести в отдельной строке "Yes", если выражение является правильным и "No" иначе.

Пример входа

```
3
([])
(([()]))
[[()[()]]()
```

Пример выхода

```
Yes
No
Yes
```

2661. Ценники

В фирму "Black Label" поступили два заказа на изготовление ценников для супермаркетов. В каждом заказе указаны количество ценников и цены, которые на них должны быть напечатаны. Вывести по одному разу все цены, которые будут напечатаны на ценниках при выполнении этих двух заказов.

Вход. В первой строке задано количество ценников n первого супермаркета. Во второй строке заданы цены, разделенные пробелами, которые должны быть указаны на ценниках. В третьей строке содержится количество ценников m второго супермаркета. В четвертой строке заданы цены, которые должны быть указаны в ценниках для второго супермаркета. Все заданные числа целые и не превышают 10^9.

Выход. Требуется вывести значения, которые будут напечатаны на ценниках – каждое по одному разу. Выводить ценники следует в возрастающем порядке.

Пример входа

```
5
100 25 300 400 12000
4
10 25 25 500
```

Пример выхода

```
10 25 100 300 400 500 12000
```


3004. Очередь

В цивилизованных странах на железнодорожном вокзале работают k касс, однако очередь в них всего одна. Обслуживание происходит следующим образом. Изначально, когда все кассы свободны, первые k человек из очереди подходят к кассам. Остальные ждут своей очереди. Как только кто-нибудь будет обслужен и соответствующая касса освободится, следующий человек из очереди подходит к этой кассе. Так продолжается до тех пор, пока не будут обслужены все клиенты.

Определите время, за которое будут обслужены все клиенты.

Вход. В первой строке находится два целых числа: размер очереди n и количество касс k ($1 \le n \le 10^5$, $1 \le k \le 10^4$). Во второй строке задаются n натуральных чисел. i-ое число определяет время t_i ($1 \le t_i \le 10^5$), которое требуется для того, чтобы обслужить i-го клиента из очереди.

Выход. Выведите одно число – время, за которое будет обслужена заданная очередь.

Пример входа

```
7 3
1 2 3 4 5 3 1
```

Пример выхода

```
7
```

АНАЛИЗ ЗАДАЧ

291. Парковка

При помощи структур данных промоделируем процесс движения автомобилей. Информацию о пустых парковочных местах будем хранить во множестве s. Таким образом поиск свободной парковки с наименьшим номером будет производиться за $O(\log_2 n)$. Для каждой машины при помощи структуры map запомним парковочное место, на которое она была поставлена. При выезде машины за $O(\log_2 n)$ можно будет найти номер парковки, которую она освобождает.

Поскольку въезд и выезд машины из парковки моделируется за $O(\log_2 n)$, то общее время работы алгоритма составит $O(m\log_2 n)$.

555. Ближайшие точки

В задаче достаточно отсортировать входные координаты x_i и найти нименьшее значение среди их соседних разностей. При этом необходимо вместе с абсциссой запоминать номер точки. Это можно сделать при помощи множества пар

```
set<pair<int, int> > s;
```

Первым элементом пары является абсцисса x_i, вторым – номер точки.

693. Минимум в стеке

В задаче достаточно промоделировать работу со стеком. Если $x_i \bmod 5 \geq 2$, то заносить в стек будем не само очередное значение x_i, а минимум вершины стека и значения x_i. Если стек пустой, то заносим в него x_i. При $x_i \bmod 5 < 2$ удаляем число из вершины стека. При таком подходе обработки входных данных на вершине стека всегда будет находиться минимальное значение среди всех обработанных, но на данный момент еще не удаленных значений x_i.

После обработки очередного значения x_i к результирующей сумме *res* необходимо добавить значение, хранящееся в вершине стека.

694. Минимум в очереди

В задаче следует промоделировать работу очереди value. Однако для того чтобы за O(1) отвечать на вопрос о минимальном элементе в очереди, заведем еще и очередь минимальных элементов mn. При поступлении или удалении элемента работу очереди value моделируем обыкновенным образом. Опишем работу очереди mn:

- При поступлении элемента x будем удалять из хвоста очереди mn элементы до тех пор, пока они будут больше x. Затем заносим x в хвост mn.
- При удалении элемента будем удалять голову очереди mn лишь в том случае, если она совпадает с удаляемым элементом.

При такой работе с очередью mn текущий наименьший элемент очереди value будет всегда находиться в голове очереди mn. А элементы очереди mn будут всегда отсортированы по возрастанию.

Если при решении задачи использовать структуру данных deque из стандартной библиотеки шаблонов, то можно получить Time Limit. Следует промоделировать работу обоих очередей через статические массивы.

Пример. Промоделируем работу очередей, например, на следующих командах. Вставка элемента происходит в конец очереди, а удаление из головы.

команда	очередь value	очередь mn
ENQUEUE (5)	(5)	(5)
ENQUEUE (7)	(5, 7)	(5, 7)
ENQUEUE (6)	(5, 7, 6)	(5, 6)
DEQUEUE	(7, 6)	(6)
ENQUEUE (4)	(7, 6, 4)	(4)
ENQUEUE (9)	(7, 6, 4, 9)	(4, 9)
DEQUEUE	(6, 4, 9)	(4, 9)
ENQUEUE (2)	(6, 4, 9, 2)	(2)

790. Двойная очередь

Приоритеты клиентов будем хранить во множестве s. Тогда клиент с наименьшим приоритетом будет находиться в начале множества, а с наибольшим в конце. Поскольку приоритеты всех клиентов разные, можно установить взаимно однозначное соответствие между приоритетом и номером клиента. Совершим это при помощи целочисленного массива client длины 10000001. Если p – приоритет клиента, то его номер будем хранить в client[p].

Для решения задачи достаточно промоделировать процесс обслуживания клиентов.

1211. Бесконечная последовательность

Вычислить все значения A_i ($i = 0, 1, \ldots, n$) последовательности невозможно при помощи массива из-за ограничения $n \leq 10^{12}$. Для запоминания результатов будем использовать структуру map: значение m[i] будет хранить A_i. Находим значение A_n, запоминая промежуточные результаты.

1225. Черный ящик

Задача красиво решается при помощи структуры данных multiset. Занесем в пустое мультимножество s макимальный элемент (например 2000000000) и установим на него указатель *iter*. Заносим в мультимножество числа последовательности A[i]. Если текущее заносимое число A[i] меньше того, на которое указывает *iter*, то уменьшаем *iter* на 1. Иначе *iter* остается без изменения. С выводом каждого числа u[i] увеличиваем *iter* на 1. Таким образом *iter* постоянно указывает на элемент, возвращаемый при операции get.

1226. Обмен иностранцами

В задаче достаточно для каждой пары (A, B) поставить в соответствие пару (B, A). Входные пары могут повторяться. Для хранения данных будем использовать мультимножество. Для каждой входной пары (A, B) будем искать в мультимножестве пару (B, A). Если таковая имеется, то удалим ее. Иначе внесем в мультимножество пару (A, B). Если в конце обработки

данных мультимножество будет пустым, то это будет означать что для каждой пары (A, B) была найдена пара (B, A), и программу обмена студентами можно выполнить.

Пример. В первом тесте для каждой пары (A, B) найдется соответствующая пара (B, A). Во втором тесте ни одно желание студента не может быть выполнено.

1227. Первый словарь Энди

Читаем текст, выделяем из него слова, состоящие из символов латинского алфавита, и заносим их в переменную типа множество (set). Слова автоматически сортируются. Если некоторое слово в тексте повторяется, то во множество оно будет занесено только один раз. Далее при помощи итератора последовательно выводим слова. Если в последовательности букв встречается апостроф (например andy's), то такую последовательность считаем как два разных слова: andy и s.

1228. Сложить все

Каждый раз следует складывать два наименьших числа. Тогда суммарная стоимость сложения всех n чисел будет наименьшей.

1776. Рельсы

Вагоны, которые будут заезжать на станцию-тупик, будем хранить в стеке s. На стороне A вагоны находятся в последовательности 1, 2, ..., n. Если первым на стороне B должен быть вагон k, то этого можно достичь загнав в тупик все вагоны с номерами 1, 2, ..., k, после чего перегнав вагон с номером k на сторону B. Пусть после этого вторым на стороне B должен находиться вагон с номером l. Если $l < k$, то его можно перевезти в направлении B только если он на данный момент находится на вершине стека s (иначе требуемая перестановка вагонов неосуществима). Если $l > k$, то перегоняем со стороны A все вагоны вплоть до l-го в стек, после чего перевозим вагон l на сторону B. Продолжаем моделировать перевозку вагонов указанным образом, пока все вагоны не будут перевезены со стороны A в сторону B.

1868. Функция

Для запоминания результатов значений функции f из-за ограничения $n \le 10^{12}$ невозможно использовать линейный массив. С этой целью будем использовать структуру данных map.

Остается написать рекурсивную реализацию функции f, запоминая промежуточные результаты.

1871. Белка и бамбук

Промоделируем процесс перекладывания желудей. Бамбук представим в виде стека, где вместо качества текущего желудя запоминается максимальное значение качества желудей, уже находящихся в бамбуке (с учетом текущего желудя). Например, пусть вершина стека содержит значение a. Тогда если нам следует положить в бамбук желудь с качеством b, то в стек будет

занесено значение $\max(a, b)$. При доставании желудя из бамбука мы можем отвечать на требуемый в условии задачи запрос за константное время – для этого достаточно вывести значение вершины стека.

Пример. Рассмотрим, как меняется состояние стека при моделировании перекладывания желудей, приведенных в примере. Жирным шрифтом в стеке выделено выводимое значение.

качество желудя	состояние стека	выводимое значение
-	(0) – инициализация	-
3	(0, 3)	-
2	(0, 3, 3)	-
4	(0, 3, 3, 4)	-
0	(0, 3, **3**)	3
4	(0, 3, 3, 4)	-
3	(0, 3, 3, 4, 4)	-
0	(0, 3, 3, **4**)	4
0	(0, 3, **3**)	3

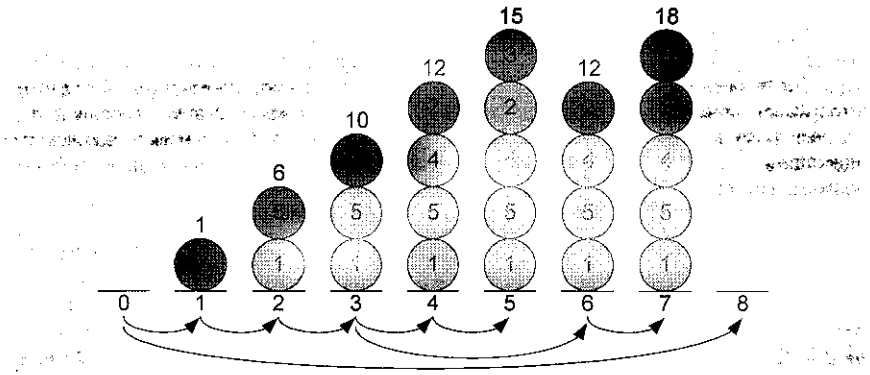
1872. Снеговики

Информацию о снеговиках будем хранить в векторе пар v. Первое число i-ой пары будет содержать номер снеговика, который был клонирован для его получения, а второе массу i-го снеговика.

- Если i-ый снеговик получается из t-го ($t < i$) добавлением шара снега, то в v[i].first следует занести значение t, а в v[i].second массу полученного снеговика, равную v[t].second + m.
- Если i-ый снеговик получается из t-го ($t < i$) удалением верхнего шара, то получится снеговик, из которого был клонирован t-ый (номер снеговика, их которого был получен t-ый, содержится в v[t].first). То есть v[i] следует положить равным v[v[t].first].

Считаем, что нулевой снеговик получен сам из себя и имеет массу 0. То есть значению v[0] присвоим пару (0, 0).

Пример. Рассмотрим приведенный в примере процесс клонирования снеговиков. Сверху над каждым снеговиком приведена его масса.



Шестой снеговик получается из пятого удалением верхнего шара. Следовательно он должен совпадать с четвертым снеговиком (из которого получился пятый). Но поскольку четвертый снеговик получен из третьего добавлением шара, то можно рассматривать шестого снеговика как такового, который получен из третьего добавлением шара с массой 2.

Восьмой снеговик получается из первого удалением верхнего шара. Он станет равным нулевому снеговику. Но поскольку считается, что нулевой снеговик получается из нулевого, то восьмой снеговик считается образованным из нулевого.

2040. Обычная перестановка

Подсчитаем количество каждой буквы, которая встречается в *a* и *b*. Находим, сколько и каких букв одновременно принадлежит строкам *a* и *b*. Выводим их (с учетом количества) в алфавитном порядке.

2479. Баланс скобок

Для решения задачи будем использовать символьный стек. Будем двигаться последовательно по символам входной строки и:

- если текущий символ является открывающейся скобкой (круглой или квадратной), то заносим его в стек.
- если текущий символ является закрывающейся скобкой, то на вершине стека должна находиться соответствующая ему открывающаяся скобка. Если это не так, или если стек пуст, то скобочное выражение не является правильным.

По окончанию обработки правильной строки стек должен оказаться пустым.

2661. Ценники

В задаче необходимо найти объединение двух множеств. Для этого достаточно занести значения всех ценников во множество. После чего достаточно вывести все элементы множества.

3004. Очередь

Будем моделировать процесс продажи билетов при помощи мультимножества *s*. Первые *k* людей подводим к свободным кассам – заносим время их обслуживания в мультимножество. При дальнейшей обработке мультимножество будет содержать *k* элементов. Каждый их них отображает момент времени, в которое соответствующая касса станет свободной и к ней сможет подойти следующий человек. Очевидно, что каждый раз новый человек должен подходить к той кассе, для которой это время минимально. Время последнего обслуженного клиента и будет искомым.

РЕАЛИЗАЦИЯ ЗАДАЧ

291. Парковка

Стоимость парковочных мест заносим в массив price, а веса автомобилей – в массив weight. Во множестве *s* будем хранить номера пустых парковочных мест. Таким образом *s*.begin() будет содержать свободное парковочное место с наименьшим номером. В отображении mp будем хранить информацию о машинах, которые поставлены на парковку. Равенство mp[*car*] = *ParkingPlace* означает, что автомобиль с номером *car* поставили на парковку номер *ParkingPlace*. Очередь машин, ждущих освобождения парковочных мест, храним в очереди d.

```
int price[101], weight[2001];
set<int> s;
map<int,int> mp;
deque<int> d;
```

Читаем входные данные.

```
scanf("%d %d",&n,&m);
```

Занесем во множество s список свободных парковочных мест. Изначально все парковки от 1 до *n* свободны.

```
for(i = 1; i <= n; i++) s.insert(i);
```

Считываем стоимость парковочных мест и веса автомобилей.

```
for(i = 1; i <= n; i++) scanf("%d",&price[i]);
for(i = 1; i <= m; i++) scanf("%d",&weight[i]);
```

Последовательно читаем и обрабатываем команды о приезде и выезде автомобилей.

```
for(res = 0, i = 1; i <= 2 * m; i++)
{
  scanf("%d",&car);
```

Автомобиль заезжает на парковку

```
  if (car > 0)
  {
```

Если множество *s* не пусто, то имеется хотя бы одно свободное парковочное место.

```
    if (!s.empty())
    {
```

Присваиваем ParkingPlace наименьший номер свободного парковочного места

```
      ParkingPlace = *s.begin();
```

К результату *res* прибавляем прибыль парковщика.

```
      res += weight[car] * price[ParkingPlace];
```

На место s.begin() только что приехала машина. Отмечаем его занятым – удаляем из множества s.

```
      s.erase(s.begin());
```

Автомобиль с номером *car* только что поставили на парковку номер *ParkingPlace*

```
      mp[car] = ParkingPlace;

    } else
    {
```

Свободных парковочных мест нет, заносим автомобиль в очередь *d*.

```
      d.push_back(car);
    }
  }
  else
```

Рассмотрим случай, когда *car* < 0. Автомобиль с номером *-car* выезжает из парковки с номером mp[*-car*]. Номер парковки, на которой находится автомобиль с номером *-car*, находим за время $O(\log_2 n)$ при помощи отображения mp.

```
  {
    ParkingPlace = mp[-car];
```

Если имеется очередь ждущих машин, то следует завести первую в очереди машину на только что освободившееся место *ParkingPlace*.

```
    if (!d.empty())
    {
      car = d[0];
      res += weight[car] * price[ParkingPlace];
      mp[car] = ParkingPlace;
      d.pop_front();
    }
    else
```

Если в очереди никого нет, то просто освобождаем парковочное место ParkingPlace.

```
      s.insert(ParkingPlace);
  }
}
```

Выводим общую прибыль парковщика.

```
printf("%d\n",res);
```

555. Ближайшие точки

Читаем входные данные. Пары (абсцисса, номер точки) автоматически сортируются по координате x_i.

```
scanf("%d",&n);
for(i = 1; i <= n; i++)
{
  scanf("%d",&x);
  s.insert(make_pair(x,i));
}
```


Вычисляем разницы между соседними абсциссами и находим среди них наименьшую в переменной *res*. Соответствующие номера точек запоминаем в переменных *a* и *b*.

```
a = b = res = 2e9;
for(iterA = iterB = s.begin(), iterB++; iterB != s.end(); iterA++, iterB++)
{
  if (((*iterB).first - (*iterA).first) < res)
  {
    res = (*iterB).first - (*iterA).first;
    a = (*iterA).second, b = (*iterB).second;
  }
}
```

Выводим ответ.

```
printf("%d\n%d %d\n",res,a,b);
```

693. Минимум в стеке

Объявим структуру данных стек.

```
stack<long long> s;
```

Читаем входные данные.

```
scanf("%lld %lld %lld %lld %lld",&n,&a,&b,&c,&x);
for(res = i = 0; i < n; i++)
{
```

В переменной *x* получаем очередной элемент x_i последовательности. В зависимости от значения *x* mod 5 или кладем в стек минимум его и значения, находящегося на вершине стека, или удаляем число из вершины стека.

```
  x = ((a*x*x + b*x + c) / 100) % 1000000;
  if (x % 5 < 2)
  {
    if (!s.empty()) s.pop();
  } else
  {
    if (s.empty()) s.push(x);
    else s.push(min(s.top(),x));
  }
```

Если стек не пустой, то прибавляем к результирующей сумме значение, хранящееся в его вершине.

```
  if (!s.empty()) res += s.top();
}
```

Выводим ответ.

```
printf("%lld\n",res);
```

694. Минимум в очереди

Очередь value будет содержать поступающие на вход числа.

```
deque<int> value, mn;
```

Читаем входные данные.

```
scanf("%d %d %d %d %d",&n,&a,&b,&c,&x);
for(i = 1; i <= n; i ++)
{
  x = ((1LL * a * x * x + 1LL * b * x + c) / 100) % 1000000LL;
  if (x % 5 < 2) // удалить из очереди
  {
```

Удаление элемента из очереди. Если она пуста, то ничего не делаем. Иначе если голова очереди совпадает с головой очереди mn (на текущий момент как раз голова очереди value являлась наименьшим элементом очереди, и при ее удалении минимальный элемент очереди изменится), то удаляем обе головы.

```
    if (!value.empty())
    {
      if (value.front() == mn.front()) mn.pop_front();
      value.pop_front();
    }
  } else  {
```

Добавляем элемент x в очередь. Удаляем из хвоста очереди mn элементы, большие x. После чего заносим x в хвост очереди mn.

```
    value.push_back(x);
    while(!mn.empty() && (x < mn.back())) mn.pop_back();
    mn.push_back(x);
  }
```

Если очередь не пуста, то ее наименьший элемент находится в голове очереди mn.

```
 if (!mn.empty()) Res += mn.front();
}
```

Выводим результат – сумму всех минимальных элементов.

```
printf("%lld\n",Res);
```

790. Двойная очередь

Приоритеты клиентов, находящихся в очереди, будем хранить во множестве s. Их соответствующие номера – в массиве client. Если на обслуживание поступает клиент с номером k и приоритетом p, то устанавливаем client[p] = k.

```
set<int> s;
int client[10000001];
```

Основная часть программы. В зависимости от типа запроса производим соответствующее действие. Продолжаем цикл, пока система обслуживания клиентов не остановится (значение *code* не станет равным 0).


```
while(scanf("%d",&code), code)
```

Заносим номер клиента k в ячейку client[p]. Заносим приоритет поступившего клиента во множество s.

```
  if (code == 1)
  {
      scanf("%d %d",&k,&p);
      client[p] = k; s.insert(p);
  } else
```

Наибольший приоритет клиента находится в конце множества s.

```
  if (code == 2)
    if (s.empty()) printf("0\n"); else
    {
      iter = s.end(); iter--;
      printf("%d\n",client[*iter]);
      s.erase(iter);
  } else
```

Наименьший приоритет клиента находится в начале множества s.

```
   if (s.empty()) printf("0\n"); else
   {
      printf("%d\n",client[*s.begin()]);
      s.erase(s.begin());
   }
```

1211. Бесконечная последовательность

Для хранения значений A_i объявим переменную m.

```
map<long long,long long> m;
```

Функция calc возвращает значение m[n].

```
long long calc(long long n, int p, int q)
{
  if (m[n]) return m[n];
  if (!n) return 1;
  return m[n] = calc(n/p,p,q) + calc(n/q,p,q);
}
```

Основная часть программы.

```
long long n, p, q, res;
scanf("%lld %lld %lld",&n,&p,&q);
res = calc(n,p,q);
printf("%lld\n",res);
```

1225. Черный ящик

Последовательность чисел A[1], ..., A[*m*], которая заносится в черный ящик, храним в массиве mas. Черный ящик хранится в переменной *s* типа мультимножество.

```
int mas[30001];
multiset<int> s;
multiset<int>::iterator iter;
```

Основной цикл программы. Обнуляем черный ящик *s*. Читаем входные данные.

```
s.clear();
scanf("%d %d",&m,&n);
for(i = 0; i < m; i++) scanf("%d",&mas[i]);
```

Занесем в *s* максимальное число для удобства его дальнейшего использования. Установим указатель *iter* на начало черного ящика – минимальный элемент, хранящийся в нем. В переменной *ptr* храним количество чисел, занесенных в черный ящик.

```
s.insert(2100000000);
iter = s.begin(); ptr = 0;
for(i = 0; i < n; i++)
{
```

Читаем значение u[*i*]. Для вывода результата на очередной запрос get, необходимо просто вывести значение, на которое указывает *iter*. Но при этом в черном ящике должно находиться u[*i*] чисел. Если до этого в ящик было занесено менее u[*i*] чисел (*ptr* < *u*), то заносим их последовательно, уменьшая на 1 указатель *iter* каждый раз когда очередное значение m[*ptr*] меньше того, на которое указывает *iter*.

```
  scanf("%d",&u);
  while(ptr < u)
  {
    s.insert(mas[ptr++]);
    if (mas[ptr-1] < *iter) iter--;
  }
  printf("%d ",*iter);iter++;
}
```

1226. Обмен иностранцами

Заведем мультимножество пар *s*. В процессе работы в нем будут храниться лишь такие пары (A, B), для которых нет соответствующей пары (B, A).

```
multiset<pair<int,int> > s;
multiset<pair<int,int> >::iterator iter;
```

Читаем количество тестов *t*.

```
scanf("%d",&t);
```

После чтения количества студентов *n* очищаем мультимножество *s*. Для каждой входной пары (A, B) проверяем, есть ли в мультимножестве пара (B, A). Если есть – то удаляем (B, A), иначе – вставляем (A, B).


```
while(t--)
{
  scanf("%d",&n); s.clear();
  for(i = 0; i < n; i++)
  {
    scanf("%d %d",&a,&b);
    if ((iter = s.find(make_pair(b,a))) == s.end()) s.insert(make_pair(a,b));
    else s.erase(iter);
  }
```

Если в конце работы мультимножество *s* пусто, то у каждой пары (A, B) существует соответствующая ей пара (B, A) и обмен студентами может быть успешно произведен.

```
  if (s.empty()) printf("YES\n");
          else printf("NO\n");
}
```

1227. Первый словарь Энди

Объявим переменную SetS типа set<string>, в которую будем заносить найденные слова. Читать входные слова будем в символьный массив *s*.

```
set<string> SetS;
set<string>::iterator iter;
char s[201];
```

Поскольку текст не обязательно начинается с буквы латинского алфавита, то пропустим начальные символы, не являющиеся буквами.

```
scanf("%[^a-zA-Z]",s);
```

Читаем слово, состоящее только из заглавных и прописных букв латинского алфавита в переменную *s*, заменяем все буквы на маленькие и заносим слово во множество SetS. Пропускаем все символы, не являющиеся буквами, читая их в массив *s*.

```
while(scanf("%[a-zA-Z]",s) == 1)
{
  for(int i = 0; i < strlen(s); i++) s[i] = tolower(s[i]);
  SetS.insert(s);
  scanf("%[^a-zA-Z]",s);
}
```

Выводим слова в алфавитном порядке.

```
for(iter = SetS.begin(); iter != SetS.end(); iter++)
  printf("%s\n",(*iter).c_str());
```

1228. Сложить все

Входные числа храним в мультимножестве *s* (числа могут повторяться). Два наименьшие числа всегда находятся в начале мультимножества.

```
multiset<int> s;
```

Читаем количество чисел *n*. Вводим последовательность слагаемых и заносим их в мультимножество *s*.

```
while(scanf("%d",&n),n)
{
  s.clear();
  for(i = 0; i < n; i++)
    scanf("%d",&num),s.insert(num);
```

В переменной *res* накапливаем стоимость сложений. Пока не осталось одно число (размер мультимножества *s* больше 1), складываем два наименьших числа и заносим их сумму в *s*. Стоимость сложения чисел *a* и *b* равно *a* + *b*.

```
  res = 0;
  while(s.size() > 1)
  {
    a = *s.begin(); s.erase(s.begin());
    b = *s.begin(); s.erase(s.begin());
    s.insert(a + b);
    res += a + b;
  }
```

Когда в мультимножестве останется одно число, выводим ответ – значение переменной *res*.

```
  printf("%d\n",res);
}
```

1776. Рельсы

Объявим стек *s*, в котором будут содержаться находящиеся на станции вагоны.

```
stack<int> s;
```

Обрабатываем последовательно входные тесты..

```
while(scanf("%d",&n),n)
{
  while(scanf("%d",&x),x)
  {
    cur = ok = 1;
```

Перед обработкой данных очередного теста очистим стек.

```
    while(!s.empty()) s.pop();
```

Обрабатываем входную последовательность.

```
    for(i = 1; i < n; i++)
    {
```

Перегоняем вагон с номером *x* на сторону B. Если вагон с номером *x* находится на стороне A, то все вагоны до *x* включительно заводим в стек.

```
      for(;cur <= x; cur++) s.push(cur);
```

Если вагон, находящийся на вершине стека, имеет номер, не равный *x*, то требуемая сортировка невозможна (в этом случае устанавливаем *ok* = 0).


```
      if (s.top() != x) ok = 0;
      s.pop();
      scanf("%d",&x);
    }
```

В зависимости от значения переменной *ok* выводим ответ.

```
    printf(ok ? "Yes\n" : "No\n");
  }
  printf("\n");
}
```

1868. Функция

Объявим отображение m. Реализуем вычисление функции f. Поскольку m[*n*] имеет тип unsigned int, то все суммирования в функции f будут выполняться по модулю 2^{32}.

```
map<unsigned long long, unsigned int> m;
unsigned int f(unsigned long long n)
{
  if (m[n] > 0) return m[n];
  if (n <= 2) return m[n] = 1;
  if (n & 1)
    return m[n] = f(6*n/7) + f(2*n/3);
  else
    return m[n] = f(n - 1) + f(n - 3);
}
```

Основная часть программы.

```
scanf("%llu",&n);
printf("%u\n",f(n));
```

1871. Белка и бамбук

Объявим стек *s*, который будет хранить максимальное значение качества желудей, уже находящихся в бамбуке (с учетом текущего желудя).

```
stack<int> s;
```

Читаем входные данные. Инициализируем стек значением 0.

```
scanf("%d",&n);
s.push(0);
while(n--)
{
  scanf("%d",&val);
```

В зависимости от значения *val* кладем желудь в бамбук или достаем его оттуда.

```
  if (val > 0) s.push(max(val,s.top()));
  else
  {
    s.pop();
    printf("%d\n",s.top());
  }
}
```

1872. Снеговики

Объявим вектор пар v, в котором будем хранить информацию о снеговиках.

```
vector<pair<int,int> > v(200010);
```

Моделируем процесс создания снеговиков как описано в анализе задачи.

```
scanf("%d",&n); sum = 0;
for(i = 1; i <= n; i++)
{
  scanf("%d %d",&t,&m);
  if (m > 0) v[i] = make_pair(t,v[t].second + m);
  else v[i] = v[v[t].first];
  sum += v[i].second;
}
```

Выводим искомую суммарную массу снеговиков.

```
printf("%lld\n",sum);
```

2040. Обычная перестановка

Входные строки считываем в массивы *a* и *b*. В отображениях *aa* и *bb* будем подсчитывать сколько и каких букв содержится в *a* и *b*.

```
char a[1010], b[1010];
map<char,int> aa, bb;
```

Для каждого теста читаем входные строки *a* и *b*. Заполняем отображения *aa* и *bb*.

```
while(gets(a))
{
  gets(b);
  aa.clear(); bb.clear();
  for(i = 0; i < strlen(a); i++) aa[a[i]]++;
  for(i = 0; i < strlen(b); i++) bb[b[i]]++;
```

Выводим общие буквы строк *a* и *b* в алфавитном порядке. При этом каждую букву выводим столько раз, сколько она встречается одновременно в *a* и *b*.

```
  for(i = 'a'; i <= 'z'; i++)
  {
    m = min(aa[i],bb[i]);
    for(j = 0; j < m; j++) printf("%c",i);
  }
  printf("\n");
}
```


2479. Баланс скобок

Входное скобочное выражение будем читать в массив *stroka*. Объявим стек *s*, используемый в задаче.

```
char stroka[150];
stack<char> s;
```

Напишем вспомогательную функцию rev. Для символа *c*, обозначающего закрывающуюся скобку, возвращаем соответствующую открывающуюся скобку. Для символа '}' возвращаем '{', для ']' возвращаем '['.

```
char rev(char c)
{
  return (c == ')') ? '(' : '[';
}
```

Основная часть программы. Читаем количество тестов *tests*. Для каждой входной строки *stroka* вычисляем ее длину *len*. Переменная *flag* станет равной 1, если входное скобочное выражение не является правильным.

```
scanf("%d\n",&tests);
while(tests--)
{
  gets(stroka); len = strlen(stroka); flag = 0;
```

Очищаем стек *s*.

```
  while(!s.empty()) s.pop();
```

Двигаемся по символам входной строки *stroka*.

```
  for(i = 0; i < len; i++)
  {
    if (flag) break;
```

Заносим в стек открывающуюся скобку.

```
    if ((stroka[i] == '{') || (stroka[i] == '[')) s.push(stroka[i]);
```

Если текущий символ stroka[*i*] – закрывающаяся скобка, то выталкиваем символ из вершины стека, который должен быть соответствующей открывающейся скобкой. Если это не так, или если стек пуст, то устанавливаем *flag* равным 1.

```
    if ((stroka[i] == '}') || (stroka[i] == ']'))
    {
      if (s.empty() || (s.top() != rev(stroka[i]))) flag = 1; else s.pop();
    }
  }
```

Если в конце обработки строки стек оказался не пустым, то входное скобочное выражение не является правильным.

```
  if (!s.empty()) flag = 1;
```

В зависимости от значения переменной *flag* выводим ответ.

```
  if (flag) printf("No\n"); else printf("Yes\n");
}
```

2661. Ценники

Значения изготавливаемых ценников будем заносить во множество s.

```
set<int> s;
```

Читаем входные данные. Заносим ценники как для первого так и для второго супермаркета во множество s.

```
scanf("%d",&n);
while(n--) scanf("%d",&val), s.insert(val);
scanf("%d",&n);
while(n--) scanf("%d",&val), s.insert(val);
```

Выводим содержимое множества s.

```
for(iter = s.begin(); iter != s.end();)
{
  printf("%d",*iter);
  iter++;
  if(iter != s.end()) printf(" ");
}
printf("\n");
```

3004. Очередь

Информацию о моментах времени, в которые происходит продажа билетов в кассах, храним в мультимножестве *s*.

```
multiset<long long> s;
```

Читаем входные данные. Время обслуживания первых *k* людей заносим в мультимножество *s*.

```
scanf("%d %d",&n,&k);
for(i = 0; i < n; i++)
{
  scanf("%d",&ti);
  if (s.size() != k) s.insert(ti);
  else
  {
```

Находим человека, который раньше всего отойдет от касс, и ставим за ним следующего человека из очереди.

```
    long long temp = *s.begin();
    s.erase(s.begin());
    s.insert(temp + ti);
  }
}
```

Наибольшее число в мультимножестве *s* равно моменту времени, в который обслужится последний клиент. Оно и будет искомым.

```
while(s.size() > 1) s.erase(s.begin());
printf("%lld\n",*s.begin());
```


Реализация через очередь с приоритетами

Заведем кучу, в корне которой находится наименьший элемент.

```
priority_queue<long long, vector<long long>, greater<long long> > pq;
```

Читаем входные данные. Время обслуживания первых *k* людей заносим в очередь *pq*.

```
scanf("%d %d",&n,&k);
for(i = 0; i < n; i++)
{
  scanf("%d",&ti);
  if (pq.size() != k) pq.push(ti);
  else
  {
```

Находим человека, который раньше всего отойдет от касс, и ставим за ним следующего человека из очереди.

```
    long long temp = pq.top(); pq.pop();
    pq.push(temp + ti);
  }
}
```

Наибольшее число в очереди *pq* равно моменту времени, в который обслужится последний клиент. Оно и будет искомым.

```
while(pq.size() > 1) pq.pop();
printf("%lld\n",pq.top());
```