# Высота дерева

Реализуйте бинарное дерево поиска для целых чисел. Программа получает на вход последовательность целых чисел и строит из них дерево. Элементы в деревья добавляются в соответствии с результатом поиска их места. Если элемент уже существует в дереве, добавлять его не надо. Балансировка дерева не производится.

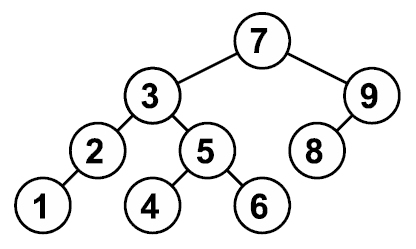
**Входные данные**

На вход программа получает последовательность натуральных чисел. Последовательность завершается числом 0, которое означает конец ввода, и добавлять его в дерево не надо.

**Выходные данные**

Выведите единственное число – высоту получившегося дерева.

Пример соответствует следующему дереву:



**Примеры**

**входные данные**

7 3 2 1 9 5 4 6 8 0

**выходные данные**

4

# Листья дерева

Реализуйте бинарное дерево поиска для целых чисел. Программа получает на вход последовательность целых чисел и строит из них дерево. Элементы в деревья добавляются в соответствии с результатом поиска их места. Если элемент уже существует в дереве, добавлять его не надо. Балансировка дерева не производится.

Последовательность завершается числом 0, которое означает конец ввода, и добавлять его в дерево не надо.

Для полученного дерева выведите список всех листьев (вершин, не имеющих потомков) в порядке возрастания.

**Входные данные**

Вводится последовательность целых чисел, оканчивающаяся нулем. Сам ноль в последовательность не входит.

**Выходные данные**

Выведите ответ на задачу.

**Примеры**

**входные данные**

7 3 2 1 9 5 4 6 8 0

**выходные данные**

1

4

6

8

# Сортировка слиянием

Отсортируйте данный массив, используя сортировку слиянием.

**Входные данные**

Первая строка входных данных содержит количество элементов в массиве *N*, *N* ≤ 105. Далее идет *N* целых чисел, не превосходящих по абсолютной величине 109.

**Выходные данные**

Выведите эти числа в порядке неубывания.

**Примеры**

**входные данные**

2

3 1

**выходные данные**

1 3

# Треугольник Паскаля

Даны два числа n и m. Создайте двумерный массив [n][m] и заполните его по следующим правилам: Числа, стоящие в строке 0 или в столбце 0 равны 1 (A[0][j]=1, A[i][0]=1). Для всех остальных элементов массива A[i][j]=A[i-1][j]+A[i][j-1], то есть каждый элемент равен сумме двух элементов, стоящих слева и сверху от него.

**Входные данные**

Программа получает на вход два числа n и m.

**Выходные данные**

Выведите данный массив.

**Примеры**

**входные данные**

3 3

**выходные данные**

1 1 1

1 2 3

1 3 6

# Обход в глубину

Дан неориентированный невзвешенный граф. Для него вам необходимо найти количество вершин, лежащих в одной компоненте связности с данной вершиной (считая эту вершину).

**Входные данные**

В первой строке входных данных содержатся два числа: N и S (1 ≤ N ≤ 100; 1 ≤ S ≤ N), где N – количество вершин графа, а S – заданная вершина. В следующих N строках записано по N чисел – матрица смежности графа, в которой 0 означает отсутствие ребра между вершинами, а 1 – его наличие. **Гарантируется,**что на главной диагонали матрицы всегда стоят нули.

**Выходные данные**

Выведите одно целое число – искомое количество вершин.

**Примеры**

**входные данные**

3 1

0 1 1

1 0 0

1 0 0

**выходные данные**

3

# Дейкстра: восстановление пути

Дан ориентированный взвешенный граф. Найдите кратчайший путь от одной заданной вершины до другой.

**Входные данные**

В первой строке содержатся три числа: N, S и F (1≤N≤100, 1≤S, F≤N), где N – количество вершин графа, S – начальная вершина, а F – конечная. В следующих N строках вводится по N чисел, не превосходящих 100, – матрица смежности графа, где -1 означает отсутствие ребра между вершинами, а любое неотрицательное число – присутствие ребра данного веса. На главной диагонали матрицы записаны нули.

**Выходные данные**

Требуется вывести последовательно все вершины одного (любого) из кратчайших путей, или одно число -1, если пути между указанными вершинами не существует. В ответе примера указано количество вершин, а не сам путь. Ваша программа должна выдавать именно путь.

**Примеры**

**входные данные**

3 2 1

0 1 1

4 0 1

2 1 0

**выходные данные**

3

# Форд-Беллман

Дан ориентированный граф, в котором могут быть кратные ребра и петли. Каждое ребро имеет вес, выражающийся целым числом (возможно, отрицательным). Гарантируется, что циклы отрицательного веса отсутствуют.

Требуется посчитать длины кратчайших путей от вершины номер 1 до всех остальных вершин.

**Входные данные**

Программа получает сначала число N (1 <= N <= 100) – количество вершин графа и число M (0 <= M <= 10000) – количество ребер. В следующих строках идет M троек чисел, описывающих ребра: начало ребра, конец ребра и вес (вес – целое число от -100 до 100).

**Выходные данные**

Программа должна вывести N чисел – расстояния от вершины номер 1 до всех вершин графа. Если пути до соответствующей вершины не существует, вместо длины пути выведите число 30000.

**Примеры**

**входные данные**

6 4

1 2 10

2 3 10

1 3 100

4 5 -10

**выходные данные**

0 10 20 30000 30000 30000

# Остовное дерево (алгоритм Прима)

Требуется найти в связном графе остовное дерево минимально веса алгоритмом Прима.

**Входные данные**

Первая строка входного файла содержит два натуральных числа n и m - количество вершин и ребер графа соответственно (1≤n≤20000, 0≤m≤100000). Следующие m строк содержат описание ребер по одному на строке. Ребро номер i описывается тремя натуральными числами bi, ei и wi - номера концов ребра и его вес соответственно (1≤bi,ei≤n, 0≤wi≤100000).

Граф является связным.

**Выходные данные**

Выведите единственное целое число - вес минимального остовного дерева.

**Примеры**

**входные данные**

4 4

1 2 1

2 3 2

3 4 5

4 1 4

**выходные данные**

7

# Остовное дерево (алгоритм Борувки)

Требуется найти в связном графе остовное дерево минимально веса алгоритмом Борувки.

**Входные данные**

Первая строка входного файла содержит два натуральных числа n и m - количество вершин и ребер графа соответственно (1≤n≤20000, 0≤m≤100000). Следующие m строк содержат описание ребер по одному на строке. Ребро номер i описывается тремя натуральными числами bi, ei и wi - номера концов ребра и его вес соответственно (1≤bi,ei≤n, 0≤wi≤100000).

Граф является связным.

**Выходные данные**

Выведите единственное целое число - вес минимального остовного дерева.

**Примеры**

**входные данные**

4 4

1 2 1

2 3 2

3 4 5

4 1 4

**выходные данные**

7

# A-функция от строки

Дана строка S, состоящая из N символов. Определим функцию **A**(i) от первых i символов этой сроки следующим образом:

**A**(i) = максимально возможному k, что равны следующие строки:

S[1]+S[2]+S[3]+…+S[k]

S[i]+S[i–1]+S[i–2]+…+S[i–k+1]

где S[i] – i-ый символ строки S, а знак + означает, что символы записываются в строчку непосредственно друг за другом.

Напишите программу, которая вычислит значения функции **A** для заданной строчки для всех возможных значений i от 1 до N.

**Входные данные**

В первой строке входного файла записано одно число N. 1≤N≤200000. Во второй строке записана строка длиной N символов, состоящая только из больших и/или маленьких латинских букв.

**Выходные данные**

В выходной файл выведите N чисел — значения функции **A**(1), **A**(2), … **A**(N).

**Примеры**

**входные данные**

5

aabaa

**выходные данные**

1 2 0 1 5

# Префикс-функция

Дана непустая строка *S*, длина которой *N* не превышает 106. Будем считать, что элементы строки нумеруются от 1 до *N*.

Для каждой позиции *i* символа в строке нас будет интересовать подстрока, заканчивающаяся в этой позиции, и совпадающая с некоторым началом всей строки. Вообще говоря, таких подстрок будет несколько, не меньше двух. Самая длинная из них имеет длину *i*, она нас интересовать не будет. А будет нас интересовать самая длинная из остальных таких подстрок (заметим, что такая подстрока всегда существует — в крайнем случае, если ничего больше не найдется, сгодится пустая подстрока).

Значением префикс-функции π[*i*] будем считать длину этой подстроки.

Требуется для всех *i* от 1 до *N* вычислить π[*i*].

**Входные данные**

Одна строка длины *N*, 0 < *N* ≤ 106, состоящая из маленьких латинских букв.

**Выходные данные**

Выведите *N* чисел — значения префикс-функции для каждой позиции, разделенные пробелом.

**Примеры**

**входные данные**

abracadabra

**выходные данные**

0 0 0 1 0 1 0 1 2 3 4

# Выпуклая оболочка

На плоскости даны N точек. Вам требуется построить выпуклую оболочку данного множества точек. Выведите два числа: периметр и площадь.

**Входные данные**

Первая строка содержит количество точек N, 1≤N≤10000. Каждая из последующих N строк содержит два целых числа – координаты xi и yi. Все числа по модулю не превосходят 104.

**Выходные данные**

Вывести два числа: периметр и площадь выпуклой оболочки.

**Примеры**

**входные данные**

4

0 0

3 4

3 1

6 0

**выходные данные**

16.0000000000

12.0000000000