

Министерство науки и высшего образования  
Российской Федерации

Московский авиационный институт  
(национальный исследовательский университет)

Институт компьютерных наук и прикладной математики

Кафедра вычислительной математики и программирования

Журнал по ознакомительной практике

Студент: Попов Н. А.

Группа: М8О-108Б-21

Оценка:

Дата:

Подпись:

Москва, 2022

# ИНСТРУКЦИЯ

## о заполнении журнала по производственной практике

Журнал по производственной практике студентов имеет единую форму для всех видов практик.

Задание в журнал вписывается руководителем практики от института в первые три-пять дней пребывания студентов на практике в соответствии с тематикой, утверждённой на кафедре до начала практики. Журнал по производственной практике является основным документом для текущего и итогового контроля выполнения заданий, требований инструкции и программы практики.

Табель прохождения практики, задание, а также технический отчёт выполняются каждым студентом самостоятельно.

Журнал заполняется студентом непрерывно в процессе прохождения всей практики и регулярно представляется для просмотра руководителям практики. Все их замечания подлежат немедленному выполнению.

В разделе «Табель прохождения практики» ежедневно должно быть указано, на каких рабочих местах и в качестве кого работал студент. Эти записи проверяются и заверяются цеховыми руководителями практики, в том числе мастерами и бригадирами. График прохождения практики заполняется в соответствии с графиком распределения студентов по рабочим местам практики, утверждённым руководителем предприятия. В разделе «Рационализаторские предложения» должно быть приведено содержание поданных в цехе рационализаторских предложений со всеми необходимыми расчётами и эскизами. Рационализаторские предложения подаются индивидуально и коллективно.

Выполнение студентом задания по общественно-политической практике заносится в раздел «Общественно-политическая практика». Выполнение работы по оказанию практической помощи предприятию (участие в выполнении спецзаданий, работа сверхурочно и т.п.) заносится в раздел журнала «Работа в помощь предприятию» с последующим письменным подтверждением записанной работы соответствующими цеховыми руководителями. Раздел «Технический отчёт по практике» должен быть заполнен

особо тщательно. Записи необходимо делать чернилами в сжатой, но вместе с тем чёткой и ясной форме и технически грамотно. Студент обязан ежедневно подробно излагать содержание работы, выполняемой за каждый день. Содержание этого раздела должно отвечать тем конкретным требованиям, которые предъявляются к техническому отчёту заданием и программой практики. Технический отчёт должен показать умение студента критически оценивать работу данного производственного участка и отразить, в какой степени студент способен применить теоретические знания для решения конкретных производственных задач.

Иллюстративный и другие материалы, использованные студентом в других разделах журнала, в техническом отчёте не должны повторяться, следует ограничиваться лишь ссылкой на него. Участие студентов в производственно-технической конференции, выступление с докладами, рационализаторские предложения и т.п. должны заноситься на свободные страницы журнала.

**Примечание.** Синьки, кальки и другие дополнения к журналу могут быть сделаны только с разрешения администрации предприятия и должны подшиваться в конце журнала.

Руководители практики от института обязаны следить за тем, чтобы каждый цеховой руководитель практики перед уходом студентов из данного цеха в другой цех вписывал в журнал студента отзывы об их работе в цехе.

Текущий контроль работы студентов осуществляется руководителями практики от института и цеховыми руководителями практики заводов. Все замечания студентам руководители делают в письменном виде на страницах журнала, ставя при этом свою подпись и дату проверки.

Результаты защиты технического отчёта заносятся в протокол и одновременно заносятся в ведомость и зачётную книжку студента.

**Примечание.** Нумерация чистых страниц журнала проставляется каждым студентом в своём журнале до начала практики.

С инструкцией о заполнении журнала ознакомлены:

«    » \_\_\_\_\_ 2021 г.  
(дата)

Студент Попов Н. А. \_\_\_\_\_  
(подпись)

## ЗАДАНИЕ

Принять участие в тренировках и соревнованиях по олимпиадному программированию для студентов первого курса в 2021/2022 учебном году: посетить и проработать установочные лекции, решать и дорешивать конкурсные задания, принять участие в разборе. Объем практики 108 часов.

Руководитель практики от института:

«    » \_\_\_\_\_ 2021 г.  
(дата)

\_\_\_\_\_  
(подпись)

## ТАБЕЛЬ ПРОХОЖДЕНИЯ ПРАКТИКИ

№	Дата	Название контеста	Время проведения	Место проведения	Решено задач	Дорешано задач	Подпись
1	30.06.2022	Основы C++	15:00 - 19:00	Дистанционно	11	1	
2	01.07.2022	Библиотека C++	12:00 - 17:00	Дистанционно	7	3	
3	02.07.2022	Динамическое программирование	12:00 - 17:00	Дистанционно	0	9	
4	04.07.2022	Префиксные суммы, сортировка событий, два указателя	12:00 - 17:00	Дистанционно	0	1	
5	05.07.2022	ДП, задача о рюкзаке	12:45 - 17:45	Дистанционно	0	0	
6	06.07.2022	Длинная арифметика	12:45 - 17:45	Дистанционно	2	0	
7	07.07.2022	Основы теории графов	12:00 - 17:00	Дистанционно	2	0	
8	08.07.2022	Кратчайшие пути во взвешенных графах	12:15 - 17:15	Дистанционно	5	0	
9	09.07.2022	Алгоритмы на строках	12:00 - 17:00	Дистанционно	1	0	
10	12.07.2022	Оформление журнала. Защита практики	9:00 - 18:00	Дистанционно			
		Итого часов	108				

## **Отзывы цеховых руководителей практики**

Принято участие в  $N$  конкурсах, прослушаны установочные лекции и разборы задач, решено  $M$  и доделано  $K$  задач конкурсов, оформлен журнал практики с электронным приложением. Задание практики выполнено. Рекомендую оценку

Тренер Инютин М. А. \_\_\_\_\_  
(подпись)

## **Работа в помощь предприятию**

Встречи с представителями ИТ-компаний, сотрудничающих с МАИ.



# ТЕХНИЧЕСКИЙ ОТЧЁТ ПО ПРАКТИКЕ

## Основы C++

### В. Развороты

ограничение по времени на тест: 1 секунда  
ограничение по памяти на тест: 256 мегабайт  
ввод: стандартный ввод  
вывод: стандартный вывод

Вам дан массив целых чисел и набор запросов. Каждый запрос задаётся парой целых чисел  $l$  и  $r$ , от вас требуют развернуть подмассив, начинающийся в позиции  $l$  и заканчивающийся в позиции  $r$ .

#### Входные данные

В первой строке вам задано единственное число  $n$  ( $1 \leq n \leq 1000$ ) — число элементов в массиве.

В следующей строке вам заданы сами элементы массива  $a_i$  ( $|a_i| \leq 10^9$ ).

В следующих строках вам задаются запросы по одному в строке в виде пары чисел разделённой пробелом  $l_i$  и  $r_i$  ( $1 \leq l_i \leq r_i \leq n$ ).

Последний запрос состоит из пары нулей, и обрабатывать его не требуется. Количество запросов не превышает 1000.

#### Выходные данные

В единственной строке выведите массив, который будет получен после обработки всех запросов.

#### Пример

входные данные	Скопировать
5 1 2 3 4 5 1 5 2 3 0 0	
выходные данные	Скопировать
5 3 4 2 1	

### Идея решения

В первую очередь мы вводим количество элементов в массиве, затем — последовательность в массив, а затем записываем порядки элементов, которые мы поменяем местами. Далее мы производим операцию swap нужных элементов (не забывая увеличивать индекс начала подпоследовательности и уменьшать индекс ее конца). Далее выводим отсортированную последовательность. Сложность алгоритма  $O(m \cdot n)$ , где  $m$  - количество запросов, а  $n$  - длина разворачиваемой подпоследовательности

### Исходный код

```
1 #include <iostream>
2 #include <vector>
3
4 using namespace std;
5
6 // vector print function
7 void print_vector(vector <long long> v)
8 {
9     for (int i = 0; i < v.size(); i++) {
```

```

10         cout << v[i];
11         if (i != v.size() - 1) {
12             cout << " ";
13         }
14     }
15 }
16
17 int main(void)
18 {
19     int n;
20     cin >> n;
21     // input vector
22     vector<long long> a(n);
23     for (int i = 0; i < n; i++) {
24         cin >> a[i];
25     }
26
27     int l, r;
28     while (1) {
29         cin >> l >> r;
30         if (l == 0 && r == 0) break;
31         l--; r--;
32         // Reversing a subarray from l to r
33         while (l < r) {
34             swap(a[l], a[r]);
35             l++;
36             r--;
37         }
38     }
39     // result output
40     print_vector(a);
41     return 0;
42 }

```

## Выводы

Задача решена.



## С. Подстроки

ограничение по времени на тест: 1 секунда  
ограничение по памяти на тест: 256 мегабайт  
ввод: стандартный ввод  
вывод: стандартный вывод

Родители решили подарить Васе на день рождения две строки, причём такие, чтобы одна была заметно меньше второй. Они знают, что Васе будет не интересно играть с этими строками, если одна будет слишком часто встречаться в другой. Ваша задача помочь родителям Васи подобрать подарок, для того чтобы определить качество подарка, они хотят знать, сколько раз меньшая строка встречается в большей.

### Входные данные

В первой строке вам даны два целых числа  $n$  ( $1 \leq n \leq 10^5$ ) и  $m$  ( $1 \leq m \leq 10^3, m \leq n$ ) — длины строк.

В следующих двух строках даны сами строки, состоящие из маленьких английских букв.

### Выходные данные

Выведите единственное число — число подстрок из первой строки, совпадающих со второй строкой.

### Примеры

<b>входные данные</b>	Скопировать
4 2 aaaa aa	
<b>выходные данные</b>	Скопировать
3	

  

<b>входные данные</b>	Скопировать
5 4 abcab abcd	
<b>выходные данные</b>	Скопировать
0	

## Идея решения

Для хранения строки и подстроки будем использовать вектор типа `char`. Далее используем алгоритм прямого поиска подстроки в строке.

Алгоритм прямого поиска подстроки в строке:

1.  $i = 1$
2. сравнить  $i$  — символ массива  $T$  с первым символом массива  $W$
3. совпадение  $\rightarrow$  сравнить вторые символы и так далее
4. несовпадение  $\rightarrow i = i + 1$  и переход на пункт 2

Сложность алгоритма  $O(n \cdot m)$ , где  $m$  - длина строки, а  $n$  - длина подстроки

## Исходный код

```
1 #include <iostream>
2 #include <vector>
3
```

```

4 using namespace std;
5
6 int main(void)
7 {
8     ios::sync_with_stdio(false);
9     cin.tie(0); cout.tie(0);
10
11     int n1, n2;
12     cin >> n1 >> n2;
13
14     // first string
15     vector<char> str1(n1);
16     // second string
17     vector<char> str2(n2);
18
19     // first string input
20     for (int i = 0; i < str1.size(); i++) {
21         cin >> str1[i];
22     }
23     // second string input
24     for (int i = 0; i < str2.size(); i++) {
25         cin >> str2[i];
26     }
27
28     int p1 = 0, ch = 0;
29     int p2 = str2.size();
30     bool flag = false;
31
32     // direct search
33     for (int i = 0; i < str1.size() - str2.size() + 1; i++) {
34         flag = true;
35         for (int j = 0; j < str2.size(); ++j)
36             if (str1[j + p1] != str2[j])
37                 flag = false;
38
39         if (flag) {
40             ch++;
41         }
42         p1++;
43     }
44     // result output
45     cout << ch << endl;
46     return 0;
47 }

```

## Выводы

Задача решена.

## Фрагмент турнирной таблицы контеста

4	Попов Николай Александрович M8O-108B-21	11	1627	+1 01:27	+1 02:32	+ 01:50	+ 01:35	+2 01:47	+ 02:04	+ 02:13	+ 02:58	+ 02:37	+ 02:51	+1 03:33	-2
---	--	----	------	-------------	-------------	------------	------------	-------------	------------	------------	------------	------------	------------	-------------	----

## А. Никогда не играйте с незнакомцами

ограничение по времени на тест: 1 секунда  
ограничение по памяти на тест: 256 мегабайт  
ввод: стандартный ввод  
вывод: стандартный вывод

Фёдор очень любит гулять неподалёку от Патриарших прудов. Во время очередной прогулки к нему подошёл незнакомец и предложил сыграть в карты. Фёдор согласился, и незнакомец объяснил ему правила.

Игра происходит колодой из карт на каждой из которых написано некоторое число — сила карты, известно, что правильный набор карт включает в себя  $N$  карт с силами  $A, A + 1, A + 2, \dots, A + N - 1$ , где  $A$  выбирается произвольным образом. На каждом ходу атакующий игрок выбирает некоторый набор карт из своей руки и выкладывает их на стол, защищающийся игрок либо выкладывает некоторый набор карт с суммарной силой строго большей чем сила карт атакующего игрока и отбивает эту атаку, либо берёт атакующий набор карт себе. После этого игроки добирают из колоды случайные карты, таким образом чтобы в руке каждого стало не менее 10 карт. Если защищающийся отбил атаку, то игроки меняются ролями и начинается новый раунд, побеждает игрок у которого в руке не осталось карт.

Фёдор крайне азартен и, к сожалению, проиграл кучу денег. Но он не готов признавать свои ошибки, по какой-то странной причине его оппонент оставил игральную колоду Фёдору, и Фёдор решил проверить, а действительно ли колода является правильной колодой для этой игры или же незнакомец его обманул.

**Входные данные**

В первой строке вам дано число  $N$  ( $1 \leq N \leq 10^5$ ) размер колоды. В следующей строке даны  $N$  чисел: силы карт в колоде  $c_i$  ( $1 \leq c_i \leq 10^9$ ).

**Выходные данные**

Если колода является корректной колодой для игры выведите «Deck looks good» без кавычек, в противном случае выведите «Scammed» без кавычек.

**Примеры**

<b>входные данные</b>	Скопировать
3 1 2 3	
<b>выходные данные</b>	Скопировать
Deck looks good	

  

<b>входные данные</b>	Скопировать
3 10 100 1000	
<b>выходные данные</b>	Скопировать
Scammed	

**Идея решения**

В первую очередь вводим последовательность в вектор. Затем мы проверяем, заряжена ли колода в киосках: если в колоде одна карта или набор карт с значениями силы как в условии задачи (разница между соседними картами равна единице), то с колодой всё в порядке. В противном случае Федю обманули.

**Исходный код**

```
1 #include <iostream>
2 #include <vector>
3 #include <algorithm>
```

```

4
5 using namespace std;
6
7 int main()
8 {
9     int n;
10    cin >> n;
11    vector<int> a(n);
12    // input
13    for (int i = 0; i < n; i++) {
14        cin >> a[i];
15    }
16
17    // sorting vector
18    sort(a.begin(), a.end());
19    for (int i = 1; i < n; i++) {
20        if (a[i] != a[i - 1] + 1) {
21            // result output
22            cout << "Scammed";
23            return 0;
24        }
25    }
26    // result output
27    cout << "Deck looks good";
28    return 0;
29 }

```

## Выводы

Задача решена.

## Фрагмент турнирной таблицы контеста

2	Попов Николай Александрович М8О-108Б-21	7	709	<span style="color: green;">+</span> 00:07	<span style="color: green;">+</span> 00:16	<span style="color: green;">+1</span> 01:17	<span style="color: green;">+</span> 00:53	<span style="color: green;">+1</span> 00:32	<span style="color: green;">+2</span> 03:38	<span style="color: blue;">-2</span>	<span style="color: blue;">-2</span>			<span style="color: green;">+2</span> 03:06	
---	--	---	-----	---	---	--	---	--	--	--------------------------------------	--------------------------------------	--	--	--	--

# Динамическое программирование

## А. Кузнечик

ограничение по времени на тест: 2 секунды  
ограничение по памяти на тест: 64 мегабайта  
ввод: стандартный ввод  
вывод: стандартный вывод

Кузнечик Пётр живёт на числовой прямой и ему нужно попасть из точки 0 в точку  $n$ , он может прыгать только в сторону увеличения координат не более чем на  $k$  шагов, то есть первый прыжок он может осуществить только в точки  $1, 2, \dots, k$ . Помогите ему определить сколькими путями он сможет это сделать, так как ответ может быть очень большой выведите его по модулю  $10^9 + 7$ .

### Входные данные

В единственной строке вам даны два числа  $n$  и  $k$  ( $1 \leq n, k \leq 2 \cdot 10^4$ ) — пункт назначения и максимальная длина прыжка кузнечика.

### Выходные данные

Выведите единственное число — ответ на задачу.

### Примеры

<b>входные данные</b>	Скопировать
10 2	
<b>выходные данные</b>	Скопировать
89	

  

<b>входные данные</b>	Скопировать
20000 2	
<b>выходные данные</b>	Скопировать
437241455	

## Идея решения

Чтобы решить эту задачу, нам нужно составить таблицу результатов при движении на определённые координаты до значения  $k$ . Значение ячейки в 0 равно единице, Для дальнейших ячеек необходимо добавить следующий шаг в цикле  $(i - j)$ , а затем взять остаток от деления на  $m$ . После цикла выводим результат из последней ячейке массива.

## Исходный код

```
1 | #include <iostream>
2 | #include <vector>
3 |
4 | using namespace std;
5 |
6 | const long long MOD = (int) 1e9 + 7;
7 |
8 | int main()
9 | {
10 |     int n, k;
11 |     cin >> n >> k;
12 |
13 |     vector<long long> dp(n + 1);
14 |     dp[0] = 1;
15 |     for (int i = 1; i <= n; i++) {
```

```

16     int j = 1;
17     while (j <= k && i - j >= 0) {
18         dp[i] += dp[i - j];
19         // Avoiding overflow of number by saving modulo number
20         dp[i] %= MOD;
21         j++;
22     }
23 }
24 // result output
25 cout << dp[n] << endl;
26 return 0;
27 }

```

## Выводы

Задача решена.

## Г. Преобразуй число

ограничение по времени на тест: 1 секунда  
ограничение по памяти на тест: 256 мегабайт  
ввод: стандартный ввод  
вывод: стандартный вывод

Имеется натуральное число  $N$ . За один ход можно вычесть из него единицу, поделить на два или поделить на три. Делить можно только нацело. Цена каждого хода — само число, над которым производится операция. Ваша задача — преобразовать число  $N$  в единицу за минимальную стоимость.

### Входные данные

В единственной строке находится натуральное число  $N$  ( $2 \leq N \leq 2 \cdot 10^7$ )

### Выходные данные

Выведите единственное число — ответ на задачу.

### Пример

входные данные	Скопировать
82	
выходные данные	Скопировать
202	

### Примечание

В первом тестовом примере следует сначала вычесть единицу, а потом делить на три. Получим  $82 + 81 + 27 + 9 + 3 = 202$ .

## Идея решения

Чтобы решить эту задачу составим таблицу из  $n$  чисел. В  $i$ -ой ячейке массива будем хранить количество способов получить из числа единицу. Заполняем массив с конца, динамически добавляя к  $i$ -ой ячейке значения массива с индексами  $i - 2$ ,  $i - 3$ ,  $i - 1$ . После заполнения массива выводим результат -  $dp[1]$ . Сложность алгоритма -  $O(n)$ .

## Исходный код

```

1 #include <iostream>
2 #include <vector>
3 #include <climits>
4

```

```

5 using namespace std;
6
7 int main()
8 {
9     // input data
10    int n;
11    cin >> n;
12
13    vector<long long> dp(n + 1, INT_MAX);
14    dp[n] = 0;
15    for (int i = n - 1; i >= 0; i--) {
16        // add to the cell the number of ways from cells i*2, i*3, i+1
17        if (dp[i + 1] + i + 1 < dp[i]) {
18            dp[i] = dp[i + 1] + i + 1;
19        }
20        if (i * 3 <= n) {
21            dp[i] = min(dp[i * 3] + i * 3, dp[i]);
22        }
23        if (i * 2 <= n) {
24            dp[i] = min(dp[i * 2] + i * 2, dp[i]);
25        }
26    }
27    // result output
28    cout << dp[1] << endl;
29    return 0;
30 }

```

## Выводы

Неправильный ответ на тесте 35. Ошибка из-за переполнения типа *int*. Изменил тип на *long long*.  
Задача решена.

## Фрагмент турнирной таблицы контеста

* Попов Николай Александрович М8О-108Б-21	9	+1	+	+2	+1	+1	+	+2	+3		+		
--	---	----	---	----	----	----	---	----	----	--	---	--	--

# Префиксные суммы, сортировка событий, два указателя

## А. Суммы подотрезков

ограничение по времени на тест: 1 секунда  
ограничение по памяти на тест: 256 мегабайт  
ввод: стандартный ввод  
вывод: стандартный вывод

Для заданного массива ответьте на запросы суммы на подотрезке массива.

### Входные данные

В первой строке дано единственное число  $N$  ( $1 \leq N \leq 2 \cdot 10^5$ ) — количество элементов в массиве. В следующей строке даны  $N$  чисел разделённых пробелом  $a_i$  ( $|a_i| \leq 10^9$ ) — элементы входного массива. В следующей строке дано число  $Q$  ( $1 \leq Q \leq 2 \cdot 10^5$ ) — количество запросов к вашей программе. В следующих  $Q$  строках заданы сами запросы в виде пар чисел разделённых пробелом  $l_i$  и  $r_i$  ( $1 \leq l_i \leq r_i \leq n$ ) — левая и правая граница запроса соответственно.

### Выходные данные

Выведите  $Q$  чисел — ответы на запросы.

### Пример

входные данные	Скопировать
3 1 -1 3 3 1 1 2 3 1 3	
выходные данные	Скопировать
1 2 3	

## Идея решения

Вводим число  $n$ . Затем создаем массив  $a$  длины  $n$  и заполняем его. Далее составляем массив префиксных сумм  $pref$  в котором будем хранить перфиксные суммы для введенного массива.

$$pref[0] = 0,$$
$$pref[j] = pref[j - 1] + a[j]$$

Затем вводим  $q$  пар чисел - левая и правая границы запроса. Для ответа на запрос пользуемся формулой:

$$\text{Сумма подотрезка } [l, r] = pref[r] - pref[l - 1]$$

Сложность заполнения масива  $pref$ :  $O(n)$

Общая сложность алгоритма:  $O(n + q)$

## Исходный код

```
1 | #include <iostream>
2 | #include <vector>
3 |
4 | using namespace std;
```



```

5
6 // function for printing vector in one line
7 void print_vector(vector<int> v) {
8     for (int i = 0; i < v.size(); i++) {
9         cout << v[i] << " ";
10    }
11    cout << endl;
12 }
13
14 int main()
15 {
16     // fast input
17     ios::sync_with_stdio(false);
18     cin.tie(0); cout.tie(0);
19     int n, q;
20     cin >> n;
21
22     vector<long long> a(n);
23     for (int i = 0; i < n; i++) {
24         cin >> a[i];
25     }
26     // vector for containing prefix sum
27     vector<long long> pref(n);
28     pref[0] = a[0];
29
30     for (int j = 0; j < n; j++) {
31         pref[j] = pref[j - 1] + a[j];
32     }
33
34     // number of queries
35     cin >> q;
36     for (int i = 0; i < q; i++) {
37         int l, r;
38         cin >> l >> r;
39         l--; r--;
40         // output query result
41         cout << pref[r] - pref[l - 1] << endl;
42     }
43     return 0;
44 }

```

## Выводы

Задача решена.

## Фрагмент турнирной таблицы конкурса

* Попов Николай Александрович М8О-1085-21	1	+2																	
--	---	----	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--

# Длинная арифметика

## В. Умножение

ограничение по времени на тест: 2 секунды  
ограничение по памяти на тест: 256 мегабайт  
ввод: стандартный ввод  
вывод: стандартный вывод

Вам заданы два целых неотрицательных числа  $A$  и  $B$ . Выведите  $A \cdot B$ .

### Входные данные

Первая строка входных данных содержит целое неотрицательное число  $A$ .

Вторая строка входных данных содержит целое неотрицательное число  $B$ .

Число разрядов в числах не превышает  $3 \cdot 10^5$ .

### Выходные данные

Выведите  $A \cdot B$ .

### Примеры

<b>входные данные</b>	Скопировать
2 2	
<b>выходные данные</b>	Скопировать
4	

  

<b>входные данные</b>	Скопировать
12345678987654321 98765432123456789	
<b>выходные данные</b>	Скопировать
1219326320073159566072245112635269	

## Идея решения

Храним числа, которые невозможно вместить в стандартный целочисленный тип в виде строк. Для умножения используем быстрое преобразование Фурье.

Сложность алгоритма:  $O(n \cdot \log_2 n)$

## Исходный код

```
1 | #include <bits/stdc++.h>
2 |
3 | using complex = std::complex<double>;
4 | using vc = std::vector<complex>;
5 |
6 | const int BASE = 10;
7 | const double PI = std::acos(-1);
8 |
9 | int rev_bits(int x, int lg_n) {
10 |     int y = 0;
11 |     for (int i = 0; i < lg_n; ++i) {
12 |         y = y << 1;
```

```

13     y ^= (x & 1);
14     x = x >> 1;
15 }
16 return y;
17 }
18
19 void fft(vc & a, bool invert)
20 {
21     int n = a.size();
22     int lg_n = 0;
23
24     while ((1 << lg_n) < n)
25         ++lg_n;
26
27     for (int i = 0; i < n; ++i)
28         if (i < rev_bits(i, lg_n))
29             swap(a[i], a[rev_bits(i, lg_n)]);
30
31     for (int layer = 1; layer <= lg_n; ++layer) {
32         int cluster = 1 << layer;
33         double phi = (2.0 * PI) / cluster;
34         if (invert)
35             phi *= -1;
36
37         complex wn = complex(std::cos(phi), std::sin(phi));
38
39         for (int i = 0; i < n; i += cluster) {
40             complex w(1);
41             for (int j = 0; j < cluster / 2; ++j) {
42                 complex u = a[i + j];
43                 complex v = a[i + j + cluster / 2] * w;
44                 a[i + j] = u + v;
45                 a[i + j + cluster / 2] = u - v;
46                 w *= wn;
47             }
48         }
49     }
50
51     if (invert)
52         for (int i = 0; i < n; ++i)
53             a[i] /= n;
54 }
55
56
57 std::string fft_mult(const std::string & a, const std::string & b) {
58     size_t max_size = std::max(a.size(), b.size());
59     size_t n = 1;
60
61     while (n < max_size)
62         n *= 2;
63
64     n *= 2;
65     vc fa(n), fb(n);
66
67     for (size_t i = 0; i < a.size(); ++i)
68         fa[a.size() - i - 1] = complex(a[i] - '0');
69
70     for (size_t i = 0; i < b.size(); ++i)
71         fb[b.size() - i - 1] = complex(b[i] - '0');
72

```

```

73     fft(fa, false);
74     fft(fb, false);
75
76     for (size_t i = 0; i < n; ++i)
77         fa[i] = fa[i] * fb[i];
78
79     fft(fa, true);
80
81     std::vector<int> res(n);
82     for (size_t i = 0; i < n; ++i)
83         res[i] = (int64_t)round(fa[i].real());
84
85     for (size_t i = 0; i < n - 1; ++i) {
86         res[i + 1] += res[i] / BASE;
87         res[i] %= BASE;
88     }
89
90     while (res.size() > 1 and res.back() == 0)
91         res.pop_back();
92
93     std::reverse(res.begin(), res.end());
94     std::string ab;
95
96     for (int64_t elem : res)
97         ab.push_back('0' + elem);
98
99     return ab;
100 }
101
102 int main(void)
103 {
104     std::ios::sync_with_stdio(false);
105     std::cin.tie(0); std::cout.tie(0);
106
107     std::string a, b;
108     std::cin >> a >> b;
109     std::string fft_res = fft_mult(a, b);
110     std::cout << fft_res << std::endl;
111
112     return 0;
113 }

```

## Выводы

Задача решена.

## Фрагмент турнирной таблицы контеста

7	Попов Николай Александрович М8О-108Б-21	2	489	+2 03:36	+	03:53	-5	
---	---	---	-----	-------------	---	-------	----	--

# Основы теории графов

## А. Поиск в глубину

ограничение по времени на тест: 1 секунда  
ограничение по памяти на тест: 64 мегабайта  
ввод: стандартный ввод  
вывод: стандартный вывод

Вам дан простой неориентированный граф, выведите для каждой вершины её номер в порядке обхода в глубину. Рёбра, исходящие из вершины, следует перебирать в порядке, в котором они заданы во входном файле.

### Входные данные

В первой строке даны  $n$ ,  $m$  и  $k$  ( $1 \leq k \leq n \leq 100000$ ,  $0 \leq m \leq \min\left(\frac{n(n-1)}{2}, 300000\right)$ ) — количество вершин и рёбер в графе и номер вершины, с которой следует начинать обход соответственно. Далее в  $m$  строках описаны рёбра графа в виде пар соединяемых ими вершин  $a$  и  $b$  ( $1 \leq a, b \leq n$ )

### Выходные данные

Выведите  $n$  чисел — номера вершин в порядке обхода в глубину от заданной вершины. Стартовая вершина имеет номер 0. Если добраться из какой-либо вершины до заданной невозможно, то вместо номера выведите  $-1$ .

### Примеры

<b>входные данные</b>	Скопировать
3 3 3 1 2 2 3 3 1	
<b>выходные данные</b>	Скопировать
2 1 0	

  

<b>входные данные</b>	Скопировать
3 1 1 1 2	
<b>выходные данные</b>	Скопировать
0 1 -1	

## Идея решения

Храним граф в виде вектора векторов целых чисел. Заполняем граф на основе входных данных. Создаем дополнительный вектор *depth* размера  $n$ , в котором будем хранить глубину каждой вершины. При обходе в глубину прежде чем переходить к следующей вершине инкрементируем текущую глубину и присваиваем ее соответствующей вершине.

Сложность алгоритма:  $O(n + m)$ , где  $n$  - число вершин, а  $m$  - число ребер графа.

## Исходный код

```
1 | #include <iostream>
2 | #include <vector>
3 |
4 | using namespace std;
5 | using graph = vector< vector<int> >;
6 |
7 | // depth-first search
8 | void dfs(int v, const graph &g, vector<int> &depth, int &curr)
```

```

9  {
10     if (depth[v] != -1)
11         return;
12     depth[v] = ++curr;
13     for (int u: g[v]) {
14         dfs(u, g, depth, curr);
15     }
16 }
17
18 int main()
19 {
20     // fast input
21     ios::sync_with_stdio(false);
22     cin.tie(0); cout.tie(0);
23
24     int n, m, k;
25     cin >> n >> m >> k;
26     k--;
27
28     graph g(n);
29     // input graph
30     for (int i = 0; i < m; i++) {
31         int a, b;
32         cin >> a >> b;
33         a--;
34         b--;
35         g[a].push_back(b);
36         g[b].push_back(a);
37     }
38
39     // vector that contains depth of each vertex
40     vector<int> depth(n, -1);
41     int curr = -1;
42
43     dfs(k, g, depth, curr);
44
45     // result output
46     for (int d : depth) {
47         cout << d << " ";
48     }
49     cout << endl;
50     return 0;
51 }

```

## Выводы

Задача решена.

## Фрагмент турнирной таблицы контеста

16	Попов Николай Александрович М8О-108Б-21	2	241	+2 01:34	+ 01:47	-2		-1			
----	---	---	-----	-------------	------------	----	--	----	--	--	--

# Кратчайшие пути во взвешенных графах

## В. Алгоритм Флойда — Уоршелла

ограничение по времени на тест: 1 секунда  
ограничение по памяти на тест: 256 мегабайт  
ввод: стандартный ввод  
вывод: стандартный вывод

Задан неориентированный взвешенный граф, вершины которого пронумерованы от 1 до  $n$ . Ваша задача найти длины кратчайших путей между всеми парами вершин.

### Входные данные

В первой строке вам дано число  $n$  ( $1 \leq n \leq 500$ ) — количество вершин в графе. В следующих  $n$  строках вам даны по  $n$  чисел  $a_{ij}$  ( $0 \leq a_{ij} \leq 10^9$ ,  $a_{ii} = 0$ ) — длины рёбер из  $i$ -й вершины в  $j$ -ю.

### Выходные данные

Выведите  $n$  строк по  $n$  чисел  $d_{ij}$  — длины кратчайших путей из вершины  $i$  в вершину  $j$ .

### Примеры

<b>входные данные</b>	Скопировать
1 0	
<b>выходные данные</b>	Скопировать
0	

  

<b>входные данные</b>	Скопировать
3 0 10 1 1 0 1 1 1 0	
<b>выходные данные</b>	Скопировать
0 2 1 1 0 1 1 1 0	

## Идея решения

Создадим специальную структуру для хранения ребер графа. В ней опишем пару вершин, которую это ребро соединяет, а также вес этого ребра.

Храним граф в виде вектора векторов ребер. Заполняем граф на основе входных данных. Создаем таблицу для хранения кратчайших расстояний между вершинами. Заполняем ее, используя алгоритм Флойда-Уоршелла. Затем выводим  $n$  строк по  $n$  чисел  $d_{ij}$  — длины кратчайших путей из вершины  $i$  в вершину  $j$ .

Временная сложность:  $O(n^3)$

Пространственная сложность:  $O(n^2)$

## Исходный код

```
1 | #include <iostream>
2 | #include <vector>
3 |
```

```

4 using namespace std;
5
6 const int64_t INF = 1e18;
7
8 // weighted edge
9 struct wedge {
10     int u, v;
11     int64_t w;
12 };
13
14 int main(void)
15 {
16     int n;
17     cin >> n;
18     vector<wedge> g;
19     // table of distances between all vertices
20     vector< vector <int64_t> > d(n, vector<int64_t> (n, INF));
21     for (int i = 0; i < n; i++) {
22         for (int j = 0; j < n; j++) {
23             if (i == j) {
24                 d[i][j] = 0;
25             }
26             int64_t w;
27             cin >> w;
28             d[i][j] = w;
29         }
30     }
31     // Floyd-Warshall algorithm
32     for (int k = 0; k < n; ++k) {
33         for (int i = 0; i < n; ++i) {
34             for (int j = 0; j < n; ++j) {
35                 d[i][j] = min(d[i][j], d[i][k] + d[k][j]);
36             }
37         }
38     }
39     // result output
40     for (int i = 0; i < n; i++) {
41         for (int j = 0; j < n; j++) {
42             cout << d[i][j] << " ";
43         }
44         cout << endl;
45     }
46     return 0;
47 }

```

## Выводы

Задача решена.

## Фрагмент турнирной таблицы контеста

5	Попов Николай Александрович М80-108Б-21	5	736	+	+	+	+	+	-4		
				01:30	01:48	02:37	03:00	03:21			



# Алгоритмы на строках

## А. Z-функция

ограничение по времени на тест: 1 секунда  
ограничение по памяти на тест: 256 мегабайт  
ввод: стандартный ввод  
вывод: стандартный вывод

Выведите  $z$ —функцию для заданной строки.

### Входные данные

В первой строке дана строка  $S$  ( $1 \leq |S| \leq 10^5$ ) состоящая из маленьких латинских букв.

### Выходные данные

В единственной строке выведите  $|S|$  чисел через пробел — значения  $z$ —функции для заданной строки.

### Примеры

<b>входные данные</b>	Скопировать
abacaba	
<b>выходные данные</b>	Скопировать
7 0 1 0 3 0 1	

  

<b>входные данные</b>	Скопировать
abababa	
<b>выходные данные</b>	Скопировать
7 0 5 0 3 0 1	

## Идея решения

В процессе вычисления  $Z$ -функции поддерживать последнее ненулевое найденное значение в виде границ отрезка  $[l; r]$ , равного соответствующему префиксу. Под “последним” значением понимается отрезок с наибольшим  $r$ .

Таким образом, в качестве начального значения  $z[i]$  можно использовать:

$$z[i] = \min(z[i - l], r - i + 1)$$

После чего запускаем наивный алгоритм, пытаясь увеличить  $z[i]$ . Это возможно, если правая граница текущего отрезка совпадения превышает  $r$ , или если  $i$  не входит в  $[l; r]$ , и вычислять  $z[i]$  необходимо с нуля.

Если в результате правая граница текущего отрезка  $(i + z[i] - 1)$  превысила  $r$ , обновляем значения  $l$  и  $r$ .

Сложность такого алгоритма равна  $O(n)$ .

## Исходный код

```
1 | #include <iostream>
2 | #include <string>
3 | #include <vector>
4 |
5 | using namespace std;
6 |
```

```

7 // z-function of string calculation function
8 vector<int> z_func(const string &s)
9 {
10     int n = s.size();
11     vector<int> z(n);
12     z[0] = n;
13     int l = 0;
14     int r = 0;
15
16     for (int i = 1; i < n; i++) {
17         if (i <= r) {
18             z[i] = min(z[i - l], r - i + 1);
19         }
20         while (i + z[i] < n and s[z[i]] == s[i + z[i]]) {
21             ++z[i];
22         }
23         if (i + z[i] >= r) {
24             l = i;
25             r = i + z[i] - 1;
26         }
27     }
28     return z;
29 }
30
31 int main(void)
32 {
33     string s;
34     cin >> s;
35     vector<int> z = z_func(s);
36     // result output
37     for (int i = 0; i < s.size(); i++) {
38         cout << z[i] << " ";
39     }
40     cout << endl;
41     return 0;
42 }

```

## Выводы

Задача решена.

## Фрагмент турнирной таблицы контеста

28	Попов Николай Александрович М8О-108Б-21	1	293	<sup>+</sup> 04:53	-1									
----	---	---	-----	-----------------------	----	--	--	--	--	--	--	--	--	--