

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ  
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «МОСКОВСКИЙ АВИАЦИОННЫЙ  
ИНСТИТУТ (НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)»

## Журнал практики

Студента Белоносова Кирилла Алексеевича (ф. и. о.)

Институт №8 «Компьютерные науки и прикладная математика»

Кафедра №805 «Математическая кибернетика»

Учебная группа 80-103Б-21

Направление подготовки (специальность) 01.03.04  
(шифр)

**Прикладная математика**

---

(название направления, специальности)

Вид практики **Ознакомительная**  
(учебной, производственной, преддипломной или другой вид практики)

Руководитель практики от МАИ

**Кудрявцева Ирина Анатольевна** \_\_\_\_\_

(фамилия, имя, отчество) (подпись)

\_\_\_\_\_/\_\_\_\_\_/ “12” июля 2022г.

(подпись студента)

(дата)

## 1. Место и сроки проведения практики

*Сроки проведения практики:*

-дата начала практики **29.06.22**

-дата окончания практики **12.07.22**

Наименование предприятия **МАИ**

Название структурного подразделения (отдел, лаборатория)

**καφ. 805**

## 2. Инструктаж по технике безопасности

\_\_\_\_\_/\_\_\_\_\_/ “ 29 ” июня \_\_\_\_\_ 2022\_ г.  
(подпись проводившего) (дата проведения)

### **3. Индивидуальное задание студенту**

1. Решить задачу «Умножение длинных чисел»
2. Решить задачу «Быстрая сортировка»
3. Решить задачу «Инвестиции»
4. Решить задачу «Поиск цикла»
5. Решить задачу «Особый префикс»

#### 4. План выполнения индивидуального задания

1. Изучение литературы и материалов связанных с необходимыми алгоритмами.
2. Решение задач с использованием выбранных алгоритмов.
3. Подготовка к защите практике. Оформление отчета.

Руководитель практики от МАИ: Кудрявцева Ирина Анатольевна. \_/\_\_\_\_\_/

Руководитель от предприятия Кудрявцева Ирина Анатольевна \_/\_\_\_\_\_/

\_\_\_\_\_/\_\_\_\_\_/ “ 29 ” \_\_\_\_\_ июня \_\_\_\_\_ 2022 г.  
(подпись студента) (дата)

### **5.Отзыв руководителя практики от предприятия**

В процессе выполнения практики студент Белоносов К.А. приобрел необходимые навыки работы с алгоритмами в олимпиадном программировании. Студент успешно использовал аппарат теории графов, дискретной математики и математического анализа.

***Материалы, изложенные в отчёте студента, полностью соответствуют индивидуальному заданию***

Руководитель от предприятия: Кудрявцева Ирина Анатольевна \_\_\_\_/\_\_\_\_\_/\_\_\_\_\_  
(фамилия, имя, отчество) (подпись)

“12” июля 2022 г.

М.П. (печать)

## 6.Отчет студента о практике

### Задание 1.

#### 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

Исходя из того, что число разрядов в числах может достигать  $3 \times 10^5$  то использование классического алгоритма умножения будет иметь асимптотику  $O(n \times m)$ , что может не влезать в временные ограничения задачи. Поэтому необходимо уменьшить асимптотику используя алгоритм быстрого преобразования Фурье. Идея алгоритма состоит в том, что мы будем представлять наши числа в виде многочленов степеней двойки и находить для Дискретное преобразование Фурье, путем разбиения нашего многочлена на 2, что как раз таки и даст логарифмическую сложность выполнения. В итоге нахождения произведения двух будет равен

$$C = \text{Inverse DFT}(\text{DFT}(A), \text{DFT}(B))$$

Сложность такого алгоритма будет уже  $O(n \times \log n)$ , но чаще всего это алгоритм применяется для больших чисел, так как для коротких чисел классический алгоритм будет быстрее.

Вывод: Данная задача как и все из блока длинной арифметики учат представлению длинных чисел в языке C++, а также работы с ними и выполнению основных операций. К концу данного блока заданий, получится готовая библиотека для работы с длинной арифметикой.

## **Исходный код**

```

1: #include <bits/stdc++.h>
2: using namespace std;
3: const int64_t INF = 1e18;
4: const double PI = std::acos(-1);
5: const int DIGIT = 1e3;
6: using pii = pair<int, int>;
7: using base = complex<double>;
8: using vc = vector<base>;
9: using vll = vector<int64_t>;
10: using ll = int64_t;
11: class bignum {
12: private:
13:     vll data;
14:     size_t size_;
15: public:
16:     const static ll DIGIT_SIZE = 3;
17:     const static ll DIGIT = 1e3;
18:     const static ll DO_FFT = 100;
19:     bignum () {
20:
21:     }
22:     bignum (ll num) {
23:         do {
24:             data.push_back(num % DIGIT);
25:             num /= DIGIT;
26:         } while(num);
27:         reverse(data.begin(), data.end());
28:         size_ = data.size();
29:     }
30:     bignum (vll _data) {
31:         data = _data;
32:         size_ = _data.size();
33:     }
34:     string remove_zero_str(const string & s) {
35:         if (s[0] != '0') {
36:             return s;
37:         }
38:         int cnt = 0;
39:         int n = s.size() - 1;
40:         while (cnt < n && s[cnt] == '0') {
41:             ++cnt;
42:         }
43:         int no_zero = s.size() - cnt + 1;
44:         return s.substr(cnt, no_zero);
45:     }
46:     bignum(const string & s_) {
47:         string s = remove_zero_str(s_);
48:         int n = s.size();
49:         int rem = n % DIGIT_SIZE;
50:         if (rem) {
51:             data.push_back(stoll(s.substr(0, rem)));
52:         }
53:         for(int i = rem; i < n; i += DIGIT_SIZE) {
54:             data.push_back(stoll(s.substr(i, DIGIT_SIZE)));
55:         }
56:         size_ = data.size();
57:     }
58:     size_t size() const {
59:         return size_;
60:     }
61:     ll & operator [] (int i) {
62:         return data[i];
63:     }
64:     const ll & operator [] (int i) const {
65:         return data[i];
66:     }
67:     static void currying(vll & vec) {
68:         size_t n = vec.size();
69:         ll carry = 0;
70:         for (size_t i = 0; i < n; ++i) {

```



```

71:         vec[i] += carry;
72:         carry = vec[i] / DIGIT;
73:         vec[i] = vec[i] % DIGIT;
74:     }
75:     if (carry) {
76:         vec.push_back(carry);
77:     }
78: }
79: static void remove_zero(vll & vec) {
80:     while(vec.size() > 1 && vec.back() == 0) {
81:         vec.pop_back();
82:     }
83: }
84: friend bignum operator + (const bignum & lhs, const bignum & rhs) {
85:     if(rhs.size() < lhs.size()) {
86:         return rhs + lhs;
87:     }
88:     size_t min_size = min(lhs.size(), rhs.size());
89:     size_t max_size = max(lhs.size(), rhs.size());
90:     vll res(max_size);
91:     for (size_t i = 0; i < min_size; ++i) {
92:         res[i] += lhs[lhs.size() - i - 1];
93:         res[i] += rhs[rhs.size() - i - 1];
94:     }
95:     for (size_t i = min_size; i < max_size; ++i) {
96:         res[i] += rhs[rhs.size() - i - 1];
97:     }
98:     currying(res);
99:     reverse(res.begin(), res.end());
100:    return res;
101: }
102: friend bignum operator - (const bignum & lhs, const bignum & rhs) {
103:     size_t max_size = lhs.size();
104:     size_t min_size = rhs.size();
105:     vll res(lhs.data);
106:     reverse(res.begin(), res.end());
107:     for (size_t i = 0; i < min_size; ++i) {
108:         res[i] -= rhs[rhs.size() - i - 1];
109:     }
110:     ll curry = 0;
111:     for(size_t i = 0; i < max_size; ++i) {
112:         res[i] -= curry;
113:         curry = 0;
114:         if (res[i] < 0) {
115:             res[i] += DIGIT;
116:             curry = 1;
117:         }
118:     }
119:     remove_zero(res);
120:     reverse(res.begin(), res.end());
121:     return bignum(res);
122: }
123: friend istream &operator >> (istream &in, bignum &num) {
124:     string s;
125:     in >> s;
126:     num = bignum(s);
127:     return in;
128: }
129: friend ostream &operator << (ostream &out, const bignum &num) {
130:     bool zero = false;
131:     for (int64_t v : num.data) {
132:         string s = to_string(v);
133:         int zeros = DIGIT_SIZE - s.size();
134:         if (zero) {
135:             for(int i = 0; i < zeros; ++i) {
136:                 out << '0';
137:             }
138:         }
139:         out << s;
140:         zero = true;

```

```

141:     }
142:     return out;
143: }
144: friend bool operator == (const bignum & lhs, const bignum & rhs) {
145:     if(lhs.size() != rhs.size()) {
146:         return false;
147:     } else {
148:         for (size_t i = 0; i < lhs.size(); ++i) {
149:             if(lhs[i] != rhs[i]) {
150:                 return false;
151:             }
152:         }
153:         return true;
154:     }
155: }
156: static bignum mult_slow(const bignum & lhs, const bignum & rhs) {
157:     size_t max_size = max(lhs.size(), rhs.size());
158:     vll res(2 * max_size);
159:     for (size_t i = 0; i < rhs.size(); ++i) {
160:         for (size_t j = 0; j < lhs.size(); ++j) {
161:             res[i + j] += rhs[rhs.size() - i - 1] * lhs[lhs.size() - j - 1];
162:         }
163:     }
164:     currying(res);
165:     remove_zero(res);
166:     reverse(res.begin(), res.end());
167:     return bignum(res);
168: }
169: static int rev_bits(int x, int n) {
170:     int y = 0;
171:     for (int i = 0; i < n; ++i) {
172:         y <<= 1;
173:         y |= (x & 1);
174:         x = x >> 1;
175:     }
176:     return y;
177: }
178: static int calc_log2_n(int n) {
179:     int res = 0;
180:     while ((1 << res) < n) {
181:         ++res;
182:     }
183:     return res;
184: }
185: static void fft (vc & a, bool invert) {
186:     int n = a.size();
187:     int lg_n = calc_log2_n(n);
188:     for (int i = 0; i < n; ++i) {
189:         if (i < rev_bits(i, lg_n)) {
190:             swap(a[i], a[rev_bits(i, lg_n)]);
191:         }
192:     }
193:     for (int layer = 1; layer <= lg_n; ++layer) {
194:         int cluster = 1 << layer;
195:         double phi = (2.0 * PI) / cluster;
196:         if (invert) {
197:             phi *= -1;
198:         }
199:         base wn = base(cos(phi), sin(phi));
200:         for (int i = 0; i < n; i += cluster) {
201:             base w(1, 0);
202:             for (int j = 0; j < cluster / 2; ++j) {
203:                 base u = a[i + j];
204:                 base v = a[i + j + cluster / 2] * w;
205:                 a[i + j] = u + v;
206:                 a[i + j + cluster / 2] = u - v;
207:                 w *= wn;
208:             }
209:         }
210:     }

```

```

211:         if(invert) {
212:             for (int i = 0; i < n; ++i) {
213:                 a[i] /= n;
214:             }
215:         }
216:     }
217:     static vc bignum_to_vc(const bignum & num, int n) {
218:         vc res(n);
219:         for (size_t i = 0; i < num.size(); ++i) {
220:             res[i] = base(num[num.size() - i - 1]);
221:         }
222:         return res;
223:     }
224:     static bignum mult_fast(const bignum & lhs, const bignum & rhs) {
225:         size_t max_size = max(lhs.size(), rhs.size());
226:         int lg_n = calc_log2_n(max_size) + 1;
227:         int n = 1 << lg_n;
228:         vc a = bignum_to_vc(lhs, n);
229:         fft(a, false);
230:         vc b = bignum_to_vc(rhs, n);
231:         fft(b, false);
232:         for(int i = 0; i < n; ++i) {
233:             a[i] *= b[i];
234:         }
235:         fft(a, true);
236:         vll res(n);
237:         for(int i = 0; i < n; ++i) {
238:             res[i] = round(a[i].real());
239:         }
240:         currying(res);
241:         remove_zero(res);
242:         reverse(res.begin(), res.end());
243:         return bignum(res);
244:     }
245:     friend bignum operator * (const bignum & lhs, const bignum & rhs) {
246:         size_t min_size = min(lhs.size(), rhs.size());
247:         if(min_size < DO_FFT) {
248:             return mult_slow(lhs, rhs);
249:         } else {
250:             return mult_fast(lhs, rhs);
251:         }
252:     }
253: };
254: int main () {
255:     cout.precision(10);
256:     cout << fixed;
257:     ios::sync_with_stdio(false);
258:     cin.tie(0);
259:     bignum a, b;
260:     cin >> a >> b;
261:     cout << a * b << "\n";
262:     return 0;
263: };

```

## Тесты

1

Время: 0 ms, память: 0 КБ

Вердикт: ОК

Ввод

2

2

Вывод участника

4

Комментарий чекера

ok answer is '4'

2

Время: 0 ms, память: 0 КБ

Вердикт: ОК

Ввод

12345678987654321

98765432123456789

Вывод участника

1219326320073159566072245112635269

Комментарий чекера

ok answer is '1219326320073159566072245112635269'

## Задание 2.

### Н. Быстрая сортировка

ограничение по времени на тест: 2 секунды  
ограничение по памяти на тест: 128 мегабайт  
ввод: стандартный ввод  
вывод: стандартный вывод

Обратите внимание на нестандартное ограничение по памяти.

Вам задан массив целых чисел, отсортируйте его.

Так как массив который требуется отсортировать слишком большой, задаваться он будет следующим образом: вам будут даны числа  $n$ ,  $a$ ,  $m$ ,  $k$  и  $mod$ , с их помощью массив получается при помощи следующего кода:

```
vector<int> v(n);  
for (int i = 0; i < n; ++i){  
    v[i] = a;  
    a = (a * m + k) % mod;  
}
```

Из-за величины массива вывести его полностью так же невозможно поэтому для проверки корректности вашей сортировки от вас требуется вывести следующую величину посчитанную после сортировки массива:

$$\sum_{i=0}^{n-1} (i+1) \cdot v[i] \pmod{(10^9 + 7)}$$

#### Входные данные

В первой строке вам дано число  $n$ , ( $1 \leq n \leq 5 \cdot 10^7$ ) — длина массива. В следующей строке вам даны числа  $a$ ,  $m$ ,  $k$  и  $mod$ , ( $0 \leq a, m, k < mod \leq 100333$ ) — числа определяющие генерацию массива.

#### Выходные данные

Выведите единственное число — ответ на задачу.

#### Пример

<b>входные данные</b>	Скопировать
3 1 1 1 10	
<b>выходные данные</b>	Скопировать
14	

## Идея

Решение данной задачи наивным алгоритмом невозможно, так как у нас большая длина массива и такое количество чисел не сможет войти в ограничение по памяти. Заметим, что каждое наше число вычисляется по модулю  $mod \leq 100333$  то есть все наши числа не будут превосходить это число, а это значит они будут находиться в кольце. Тогда мы можем применить одну из линейных сортировок — сортировкой подсчетом, давайте просто создадим отдельный массив длины  $mod$  и посчитаем сколько раз наши числа встречаются в исходной последовательности, тогда наш массив уже будет занимать  $mod$  элементов, что уже намного лучше. Сложность такого алгоритма  $O(n)$ . Вывод: Данная задача знакомит нас с сортировкой подсчетом и её применением, приобретённые знания помогут в решении задач, когда нам известен диапазон чисел в массиве.

## Исходный код

```
1: #include <bits/stdc++.h>
2: using namespace std;
3: int square(int x1, int x2, int y1, int y2) {
4:     return abs(x1 - x2)*abs(y1 - y2);
5: }
6: int main() {
7:     ios::sync_with_stdio(false);
8:     cin.tie(0);
9:     int64_t n, a, m, k, mod;
10:    cin >> n;
11:    cin >> a >> m >> k >> mod;
12:    vector<int> c(mod);
13:    for (int i = 0; i < n; ++i){
14:        c[a]++;
15:        a = (a * m + k) % mod;
16:    }
17:    int64_t sum = 0;
18:    int64_t t = 0;
19:    for (int i = 0; i < mod; ++i) {
20:        while(c[i] != 0) {
21:            t++;
22:            c[i]--;
23:            sum = (sum + (t * i)) % (1000000007);
24:        }
25:    }
26:    cout << sum << "\n";
27: }
```

## Тесты

1

Время: 0 ms, память: 0 КБ

Вердикт: ОК

Ввод

3

1 1 1 10

Вывод участника

14

Комментарий чекера

ok 1 number(s): "14"

### Задание 3.

#### Е. Инвестиции

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

Петя — начинающий инвестор с капиталом в  $m$  бурлей (считайте, что за один бурль можно купить много шоколадок!). В Берляндии есть  $n$  предприятий, в каждое из которых можно вложить некоторую сумму денег. В течении нескольких месяцев Петя создавал модель прибыльности предприятий на основе истории за прошлые десятилетия. Теперь ему известно, какая прибыль  $c_{ij}$  ожидает инвестора при вложении  $j$  бурлей в предприятие  $i$ . Конечно же, чем больше вложения, тем больше доход, то есть  $c_{i_1} < c_{i_2} < \dots < c_{i_{m-1}} < c_{i_m}$ .

Петя хоть и опытный аналитик, но вот рассчитать свои вложения ему не под силу, поэтому он просит помочь Вас. Вам известен стартовый капитал Пети  $m$ , количество предприятий  $n$  и доходность предприятий  $c_{ij}$ . Вы должны вложить в каждое предприятие какое-то число (возможно, ноль) бурлей так, чтобы суммарно вложить  $m$  бурлей и получить максимальную прибыль.

#### Входные данные

На первой строке находится число  $T$  ( $1 \leq T \leq 10$ ) — число запросов в тесте.

Каждый тест описывается числами  $n$  и  $m$  ( $1 \leq n, m \leq 100$ ) — количество предприятий и стартовый капитал Пети.

В следующих  $n$  строках находятся числа  $c_{ij}$  ( $1 \leq c_{ij} \leq 1000$ ) — доход  $i$  предприятия при вложении в него  $j$  бурлей.

Гарантируется, что сумма  $n$  и  $m$  по всем запросам не превышает 100.

#### Выходные данные

На каждый тестовый случай выведите единственное число — ответ на задачу.

#### Пример

входные данные	Скопировать
3 4 5 8 10 11 12 18 6 9 11 13 15 3 4 7 11 18 4 6 8 13 16 4 5 8 16 24 28 32 12 22 30 35 37 10 19 26 32 36 9 17 25 32 38 3 7 3 5 9 11 17 18 21 5 8 9 15 19 21 22 4 5 11 12 18 22 24	
выходные данные	Скопировать
24 50 27	

#### Примечание

В первом тестовом случае можно распределить средства следующим образом: (1, 2, 1, 1). Получим  $8 + 9 + 3 + 4 = 24$ .

Во втором тестовом случае можно распределить средства следующим образом: (0, 2, 2, 1). Получим  $0 + 22 + 19 + 9 = 50$ .

В третьем тестовом случае можно распределить средства следующим образом: (0, 1, 6). Получим  $0 + 5 + 22 = 27$ .

## Идея

Данная задача представляет собой классическую задачу о рюкзаке, решаемую методом динамического программирования, где нам нужно провести выбор максимально эффективного вклада. Но нам нужно выбирать не и общего набора, а из  $n$  различных предприятий, тогда добавим еще один цикл, который будет искать максимум по всем этим  $n$  наборам. Тогда сложность составит  $O(n^2 \times m)$ . Вывод: Данная задача показывает применение задачи о рюкзаке, а также отрабатывает технику её использования в различных ситуациях.

## Исходный код

```
1: #include <bits/stdc++.h>
2: using namespace std;
3: const int64_t INF = 1e18;
4:
5: void solve() {
6:     int n, W;
7:     cin >> n >> W;
8:     vector<vector<int>>> c(n, vector<int>(W));
9:     vector<vector<int>>> w(n, vector<int>(W));
10:    for (int i = 0; i < n; ++i) {
11:        for (int j = 0; j < W; ++j) {
12:            cin >> c[i][j];
13:            w[i][j] = j + 1;
14:        }
15:    }
16:    cout << "\n";
17:    vector<vector<int>>> dp(n + 1, vector<int>(W + 1));
18:    for (int j = 1; j <= W; ++j) {
19:        for (int i = 1; i <= n; ++i) {
20:            dp[i][j] = dp[i - 1][j];
21:            for (int k = 1; k <= W; ++k) {
22:                int wi = w[i - 1][k - 1];
23:                int ci = c[i - 1][k - 1];
24:                if(j - wi >= 0) {
25:                    dp[i][j] = max(dp[i][j], dp[i - 1][j - wi] + ci);
26:                }
27:            }
28:        }
29:    }
30:    // for (int j = 0; j <= W; ++j) {
31:    //     for (int i = 0; i <= n; ++i) {
32:    //         cout << dp[i][j] << '\t';
33:    //     }
34:    //     cout << "\n";
35:    // }
36:    int ans = dp.back().back();
37:    cout << ans << "\n";
38: }
39: int main () {
40:     ios::sync_with_stdio(false);
41:     cin.tie(0);
42:     int t;
43:     cin >> t;
44:     for (int i = 0; i < t; ++i) {
45:         solve();
46:     }
47:     return 0;
48: }
```



## Тесты

1

Время: 0 ms, память: 8 КБ

Вердикт: ОК

Ввод

```
3
4 5
8 10 11 12 18
6 9 11 13 15
3 4 7 11 18
4 6 8 13 16
4 5
8 16 24 28 32
12 22 30 35 37
10 19 26 32 36
9 17 25 32 38
3 7
3 5 9 11 17 18 21
5 8 9 15 19 21 22
4 5 11 12 18 22 24
```

Вывод участника

24

50

27

Комментарий чекера

ok 3 number(s): "24 50 27"

## Задание 4.

### Е. Поиск цикла

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

Вам дан простой неориентированный связный граф с одним циклом, найдите вершины входящие в него.

#### Входные данные

В первой строке даны  $n$  ( $3 \leq n \leq 100000$ ) — количество вершин в графе. Далее в  $n$  строках описаны рёбра графа в виде пар соединяемых ими вершин.

#### Выходные данные

Выведите номера всех вершин, входящих в цикл, по возрастанию.

#### Примеры

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

  

<b>входные данные</b>	Скопировать
5 1 2 2 3 3 4 4 5 4 2	
<b>выходные данные</b>	Скопировать
2 3 4	

### Идея

Будем идти обходом в глубину по нашему графу и раскрашивать наши вершины:

Белый – вершина не посещена.

Серый – вершина в состоянии обхода.

Черный – обход для вершины закончен.

Если во время обхода мы нашли серую вершины, значит мы нашли цикл.

Сложность алгоритма равна сложности DFS –  $O(n + m)$ , где  $n$  – количество вершин,  $m$  – количество ребер.

Вывод: Эта задача отрабатывает алгоритм поиска в глубину и учит находить циклы, что будет очень полезным, для нахождения оптимального пути, не содержащего циклов.

## Исходный код

```
1: #include <bits/stdc++.h>
2: using namespace std;
3: const int INF = 1e9;
4: const int WHITE = 0;
5: const int GRAY = 1;
6: const int BLACK = 2;
7: using pii = pair<int, int>;
8: using graph = vector< vector<int>>>;
9: bool dfs(int u, int parent, const graph & g, vector<int> & colour, vector<int> & path, vector<int> &
ans) {
10:     if(colour[u] == BLACK) {
11:         return false;
12:     }
13:     if(colour[u] == GRAY ) {
14:         return true;
15:     }
16:     colour[u] = GRAY;
17:     path.push_back(u);
18:     for(int v : g[u]) {
19:         if (v != parent) {
20:             bool found_cycle = dfs(v, u, g, colour, path, ans);
21:             if(found_cycle) {
22:                 if(ans.empty()) {
23:                     int i = path.size() - 1;
24:                     while(path[i] != v) {
25:                         ans.push_back(path[i]);
26:                         --i;
27:                     }
28:                     ans.push_back(v);
29:                 }
30:                 return true;
31:             }
32:         }
33:     }
34:     colour[u] = BLACK;
35:     path.pop_back();
36:     return false;
37: }
38: int main() {
39:     ios::sync_with_stdio(false);
40:     cin.tie(0);
41:     int n;
42:     cin >> n;
43:     graph g(n);
44:     for (int i = 0; i < n; ++i) {
45:         int u, v;
46:         cin >> u >> v;
47:         --u;
48:         --v;
49:         g[u].push_back(v);
50:         g[v].push_back(u);
51:     }
52:     for(int i = 0; i < n; ++i) {
53:         sort(g[i].begin(), g[i].end());
54:     }
55:     vector<int> colour(n, WHITE);
56:     vector<int> path;
57:     vector<int> ans;
58:     dfs(0, 0, g, colour, path, ans);
59:     reverse(ans.begin(), ans.end());
60:     sort(ans.begin(), ans.end());
61:     for (int v : ans) {
62:         cout << v + 1 << " ";
63:     }
64: }
```

# Тесты

**1**

**Время:** 0 ms, **память:** 0 КБ

**Вердикт:** ОК

**Ввод**  
3  
1 2  
2 3  
3 1

**Вывод участника**  
1 2 3

**Комментарий чекера**  
ok 3 number(s): "1 2 3"

**2**

**Время:** 0 ms, **память:** 8 КБ

**Вердикт:** ОК

**Ввод**  
5  
1 2  
2 3  
3 4  
4 5  
4 2

**Вывод участника**  
2 3 4

**Комментарий чекера**  
ok 3 number(s): "2 3 4"

## Задание 5.

### Е. Особый префикс

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

Дана строка  $s$ . Найдите длину наибольшего префикса, который является палиндромом.

Напомним, что палиндром — это строка, которая читается одинаково справа налево и слева направо.

#### Входные данные

На вход подаётся единственная строка  $s$  ( $1 \leq |s| \leq 10^6$ ).

#### Выходные данные

Выведите единственное число — длину наибольшего префикса  $s$ , который является палиндромом.

#### Примеры

<b>входные данные</b>	Скопировать
ababacaba	
<b>выходные данные</b>	Скопировать
5	

  

<b>входные данные</b>	Скопировать
ababacababa	
<b>выходные данные</b>	Скопировать
11	

#### Идея

Решить данное задание можно с использованием Z-Функции. Воспользуемся свойством палиндрома, и запишем новую строку  $T = s + \# + s.reverse()$ , тогда нетрудно догадаться, что значение Z-Функции будет максимально в тех позициях, которые являются палиндромами, но палиндромы могут быть и с конца, тогда индекс нашего элемента должен удовлетворять свойству  $z[i] + i = n$ . Сложность алгоритма равна сложности вычисления Z-Функции, а именно  $O(n)$ .

Вывод: Данная задача отрабатывает алгоритм использования Z-Функции, а также показывает её мощь при работе с различными видами строк.

## Исходный код

```
1: #include <bits/stdc++.h>
2: using namespace std;
3: vector<int> z_func(const string & s) {
4:     int n = s.size();
5:     vector<int> z(n);
6:     z[0] = s.size();
7:     int l = 0, r = 0;
8:     for (int i = 1; i < n; ++i) {
9:         if (i <= r) {
10:            z[i] = min(z[i - l], r - i + 1);
11:        }
12:        while(i + z[i] < n && s[z[i]] == s[i + z[i]]) {
13:            ++z[i];
14:        }
15:        if (i + z[i] >= r) {
16:            l = i;
17:            r = l + z[i] - 1;
18:        }
19:    }
20:    return z;
21: }
22: int main() {
23:     ios::sync_with_stdio(false);
24:     cin.tie(0);
25:     string s;
26:     cin >> s;
27:     string sr = s;
28:     reverse(sr.begin(), sr.end());
29:     string buf = s + '#' + sr;
30:     vector<int> z = z_func(buf);
31:     int ans = 0;
32:     for (int64_t i = (int64_t)s.size() + 1; i < 2*(int64_t)s.size() + 1; ++i) {
33:         if(z[i] > ans) {
34:             ans = z[i];
35:         }
36:     }
37:     s = s.substr(0, ans);
38:     sr = sr.substr(sr.size() - ans, ans);
39:     buf = s + '#' + sr;
40:     z = z_func(buf);
41:     ans = 0;
42:     for (int64_t i = (int64_t)s.size() + 1; i < 2*(int64_t)s.size() + 1; ++i) {
43:         if(z[i] > ans) {
44:             ans = z[i];
45:         }
46:     }
47:     cout << ans << "\n";
48:     return 0;
49: }
```

## Тесты

1

Время: 0 ms, память: 0 КБ

Вердикт: ОК

Ввод

ababacaba

Вывод участника

5

Комментарий чекера

ok 1 number(s): "5"

2

Время: 0 ms, память: 4 КБ

Вердикт: ОК

Ввод

ababacababa

Вывод участника

11

Комментарий чекера

ok 1 number(s): "11"