**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)*

***каф. 805***

1. **Инструктаж по технике безопасности**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/ “\_\_29\_\_”\_\_\_июня\_\_\_\_\_\_\_\_\_ 2022\_ г.

*(подпись проводившего)* *(дата проведения)*

1. **Индивидуальное задание студенту**
2. Решить задачу «Умножение длинных чисел»
3. Решить задачу «Быстрая сортировка»
4. Решить задачу «Инвестиции»
5. Решить задачу «Поиск цикла»
6. 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 *\_\_\_\_/\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/*

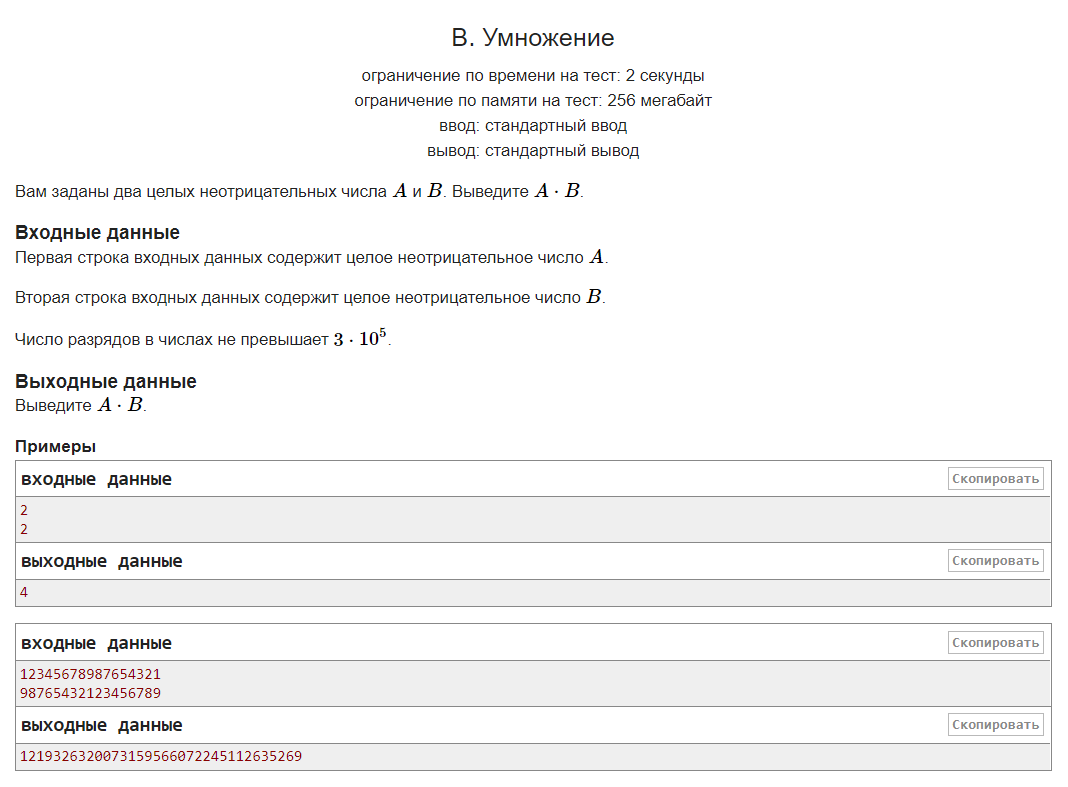
*(фамилия, имя, отчество) (подпись)*

“12” июля 2022 г.

М.П. (печать)

**6.Отчет студента о практике**

**Задание 1.**

****

**Идея решения**

Исходя из того, что число разрядов в числах может достигать 3 × 105 то использование классического алгоритма умножения будет иметь асимптотику О(n × 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 = Inverse DFT(DFT(A), DFT(B))

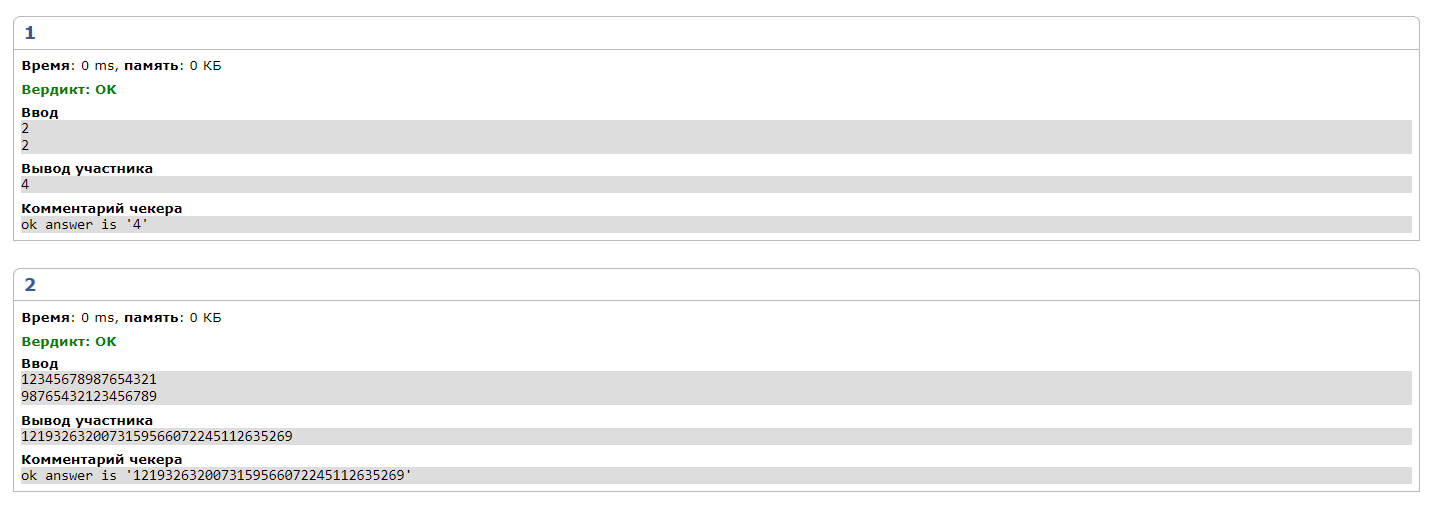
Сложность такого алгоритма будет уже O(n×logn), но чаще всего это алгоритм применяется для больших чисел, так как для коротких чисел классический алгоритм будет быстрее.

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.

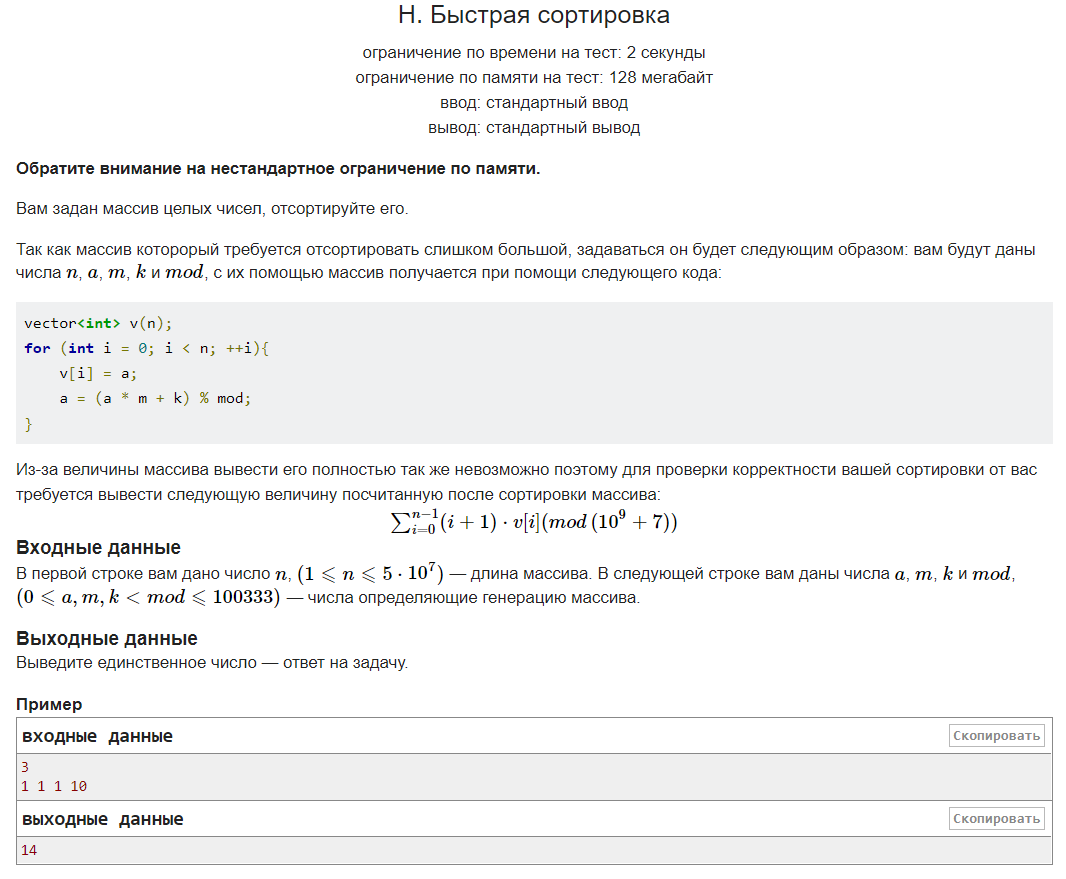
**Исходный код**

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. bignum (ll num) {
22. do {
23. data.push\_back(num % DIGIT);
24. num /= DIGIT;
25. } while(num);
26. reverse(data.begin(), data.end());
27. size\_ = data.size();
28. }
29. bignum (vll \_data) {
30. data = \_data;
31. size\_ = \_data.size();
32. }
33. string remove\_zero\_str(const string & s) {
34. if (s[0] != '0') {
35. return s;
36. }
37. int cnt = 0;
38. int n = s.size() - 1;
39. while (cnt < n && s[cnt] == '0') {
40. ++cnt;
41. }
42. int no\_zero = s.size() - cnt + 1;
43. return s.substr(cnt, no\_zero);
44. }
45. bignum(const string & s\_) {
46. string s = remove\_zero\_str(s\_);
47. int n = s.size();
48. int rem = n % DIGIT\_SIZE;
49. if (rem) {
50. data.push\_back(stoll(s.substr(0, rem)));
51. }
52. for(int i = rem; i < n; i += DIGIT\_SIZE) {
53. data.push\_back(stoll(s.substr(i, DIGIT\_SIZE)));
54. }
55. size\_ = data.size();
56. }
57. size\_t size() const {
58. return size\_;
59. }
60. ll & operator [] (int i) {
61. return data[i];
62. }
63. const ll & operator [] (int i) const{
64. return data[i];
65. }
66. static void currying(vll & vec) {
67. size\_t n = vec.size();
68. ll carry = 0;
69. for (size\_t i = 0; i < n; ++i) {
70. vec[i] += carry;
71. carry = vec[i] / DIGIT;
72. vec[i] = vec[i] % DIGIT;
73. }
74. if (carry) {
75. vec.push\_back(carry);
76. }
77. }
78. static void remove\_zero(vll & vec) {
79. while(vec.size() > 1 && vec.back() == 0) {
80. vec.pop\_back();
81. }
82. }
83. friend bignum operator + (const bignum & lhs, const bignum & rhs) {
84. if(rhs.size() < lhs.size()) {
85. return rhs + lhs;
86. }
87. size\_t min\_size = min(lhs.size(), rhs.size());
88. size\_t max\_size = max(lhs.size(), rhs.size());
89. vll res(max\_size);
90. for (size\_t i = 0; i < min\_size; ++i) {
91. res[i] += lhs[lhs.size() - i - 1];
92. res[i] += rhs[rhs.size() - i - 1];
93. }
94. for (size\_t i = min\_size; i < max\_size; ++i) {
95. res[i] += rhs[rhs.size() - i - 1];
96. }
97. currying(res);
98. reverse(res.begin(), res.end());
99. return res;
100. }
101. friend bignum operator - (const bignum & lhs, const bignum & rhs) {
102. size\_t max\_size = lhs.size();
103. size\_t min\_size = rhs.size();
104. vll res(lhs.data);
105. reverse(res.begin(), res.end());
106. for (size\_t i = 0; i < min\_size; ++i) {
107. res[i] -= rhs[rhs.size() - i - 1];
108. }
109. ll curry = 0;
110. for(size\_t i = 0; i < max\_size; ++i) {
111. res[i] -= curry;
112. curry = 0;
113. if (res[i] < 0) {
114. res[i] += DIGIT;
115. curry = 1;
116. }
117. }
118. remove\_zero(res);
119. reverse(res.begin(), res.end());
120. return bignum(res);
121. }
122. friend istream &operator >> (istream &in, bignum &num) {
123. string s;
124. in >> s;
125. num = bignum(s);
126. return in;
127. }
128. friend ostream &operator << (ostream &out, const bignum &num) {
129. bool zero = false;
130. for (int64\_t v : num.data) {
131. string s = to\_string(v);
132. int zeros = DIGIT\_SIZE - s.size();
133. if (zero) {
134. for(int i = 0; i < zeros; ++i) {
135. out << '0';
136. }
137. }
138. out << s;
139. zero = true;
140. }
141. return out;
142. }
143. friend bool operator == (const bignum & lhs, const bignum & rhs) {
144. if(lhs.size() != rhs.size()) {
145. return false;
146. } else {
147. for (size\_t i = 0; i < lhs.size(); ++i) {
148. if(lhs[i] != rhs[i]) {
149. return false;
150. }
151. }
152. return true;
153. }
154. }
155. static bignum mult\_slow(const bignum & lhs, const bignum & rhs) {
156. size\_t max\_size = max(lhs.size(), rhs.size());
157. vll res(2 \* max\_size);
158. for (size\_t i = 0; i < rhs.size(); ++i) {
159. for (size\_t j = 0; j < lhs.size(); ++j) {
160. res[i + j] += rhs[rhs.size() - i - 1] \* lhs[lhs.size() - j - 1];
161. }
162. }
163. currying(res);
164. remove\_zero(res);
165. reverse(res.begin(), res.end());
166. return bignum(res);
167. }
168. static int rev\_bits(int x, int n) {
169. int y = 0;
170. for (int i = 0; i < n; ++i) {
171. y <<= 1;
172. y |= (x & 1);
173. x = x >> 1;
174. }
175. return y;
176. }
177. static int calc\_log2\_n(int n) {
178. int res = 0;
179. while ((1 << res) < n) {
180. ++res;
181. }
182. return res;
183. }
184. static void fft (vc & a, bool invert) {
185. int n = a.size();
186. int lg\_n = calc\_log2\_n(n);
187. for (int i = 0; i < n; ++i) {
188. if (i < rev\_bits(i, lg\_n)) {
189. swap(a[i], a[rev\_bits(i, lg\_n)]);
190. }
191. }
192. for (int layer = 1; layer <= lg\_n; ++layer) {
193. int cluster = 1 << layer;
194. double phi = (2.0 \* PI) / cluster;
195. if (invert) {
196. phi \*= -1;
197. }
198. base wn = base(cos(phi), sin(phi));
199. for (int i = 0; i < n; i += cluster) {
200. base w(1, 0);
201. for (int j = 0; j < cluster / 2; ++j) {
202. base u = a[i + j];
203. base v = a[i + j + cluster / 2] \* w;
204. a[i + j] = u + v;
205. a[i + j + cluster / 2] = u - v;
206. w \*= wn;
207. }
208. }
209. }
210. if(invert) {
211. for (int i = 0; i < n; ++i) {
212. a[i] /= n;
213. }
214. }
215. }
216. static vc bignum\_to\_vc(const bignum & num, int n) {
217. vc res(n);
218. for (size\_t i = 0; i < num.size(); ++i) {
219. res[i] = base(num[num.size() - i - 1]);
220. }
221. return res;
222. }
223. static bignum mult\_fast(const bignum & lhs, const bignum & rhs) {
224. size\_t max\_size = max(lhs.size(), rhs.size());
225. int lg\_n = calc\_log2\_n(max\_size) + 1;
226. int n = 1 << lg\_n;
227. vc a = bignum\_to\_vc(lhs, n);
228. fft(a, false);
229. vc b = bignum\_to\_vc(rhs, n);
230. fft(b, false);
231. for(int i = 0; i < n; ++i) {
232. a[i] \*= b[i];
233. }
234. fft(a, true);
235. vll res(n);
236. for(int i = 0; i < n; ++i) {
237. res[i] = round(a[i].real());
238. }
239. currying(res);
240. remove\_zero(res);
241. reverse(res.begin(), res.end());
242. return bignum(res);
243. }
244. friend bignum operator \* (const bignum & lhs, const bignum & rhs) {
245. size\_t min\_size = min(lhs.size(), rhs.size());
246. if(min\_size < DO\_FFT) {
247. return mult\_slow(lhs, rhs);
248. } else {
249. return mult\_fast(lhs, rhs);
250. }
251. }
252. };
253. int main () {
254. cout.precision(10);
255. cout << fixed;
256. ios::sync\_with\_stdio(false);
257. cin.tie(0);
258. bignum a, b;
259. cin >> a >> b;
260. cout << a \* b << "\n";
261. return 0;
262. };

**Тесты**

****

**Задание 2.**

****

**Идея**

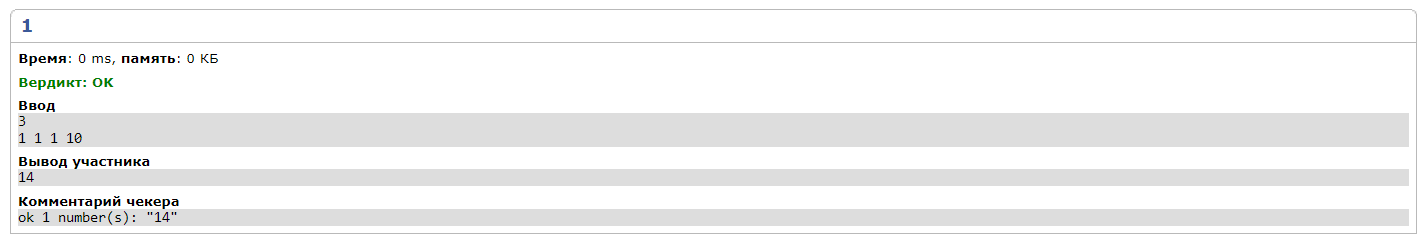
Решение данной задачи наивным алгоритмом невозможно, так как у нас большая длина массива и такое количество чисел не сможет войти в ограничение по памяти. Заметим, что каждое наше число вычисляется по модулю mod ≤ 100333 то есть все наши числа не будут превосходить это число, а это значит они будут находиться в кольце. Тогда мы можем применить одну из линейных сортировок – сортирвкой подсчетом, давайте просто создадим отдельный массив длины mod и посчитаем сколько раз наши числа встречаются в исходной последовательности, тогда наш массив уже будет занимат mod элеменетов, 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.

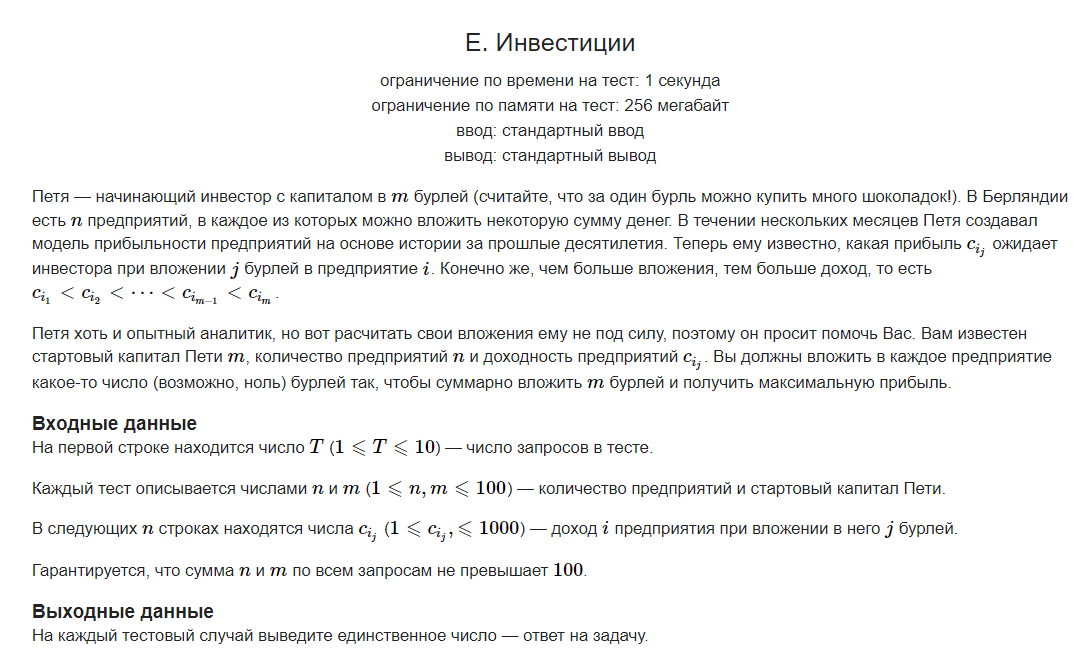
**Исходный код**

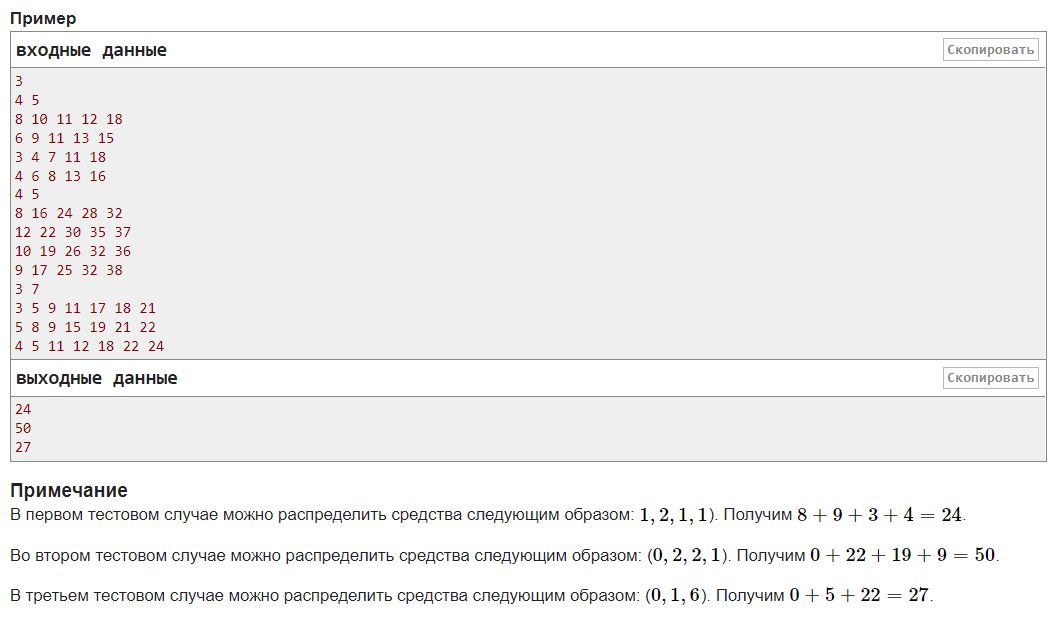
1. #include <bits/stdc++.h>
2. using namespace std;
3. int square(int x1, int x2, int y1, int y2) {
4. return abs(x1 - x2)\*abs(y1 - y2);
5. }
6. int main() {
7. ios::sync\_with\_stdio(false);
8. cin.tie(0);
9. int64\_t n, a, m, k, mod;
10. cin >> n;
11. cin >> a >> m >> k >> mod;
12. vector<int> c(mod);
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**

****

**Задание 3.**

****



**Идея**

Данная задача представляет собой классическую задачу о рюкзаке, решающуюся методом динамического программирования, где нам нужно провести выбор максимально эффективного вклада. Но нам нужно выбирать не и общего набора, а из n различный предприятий, тогда добавим еще один цикл, который будет искать максимум по всем этим n наборам. Тогда сложность составит O(n2 × 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. void solve() {
5. int n, W;
6. cin >> n >> W;
7. vector<vector<int>> c(n, vector<int>(W));
8. vector<vector<int>> w(n, vector<int>(W));
9. for (int i = 0; i < n; ++i) {
10. for (int j = 0; j < W; ++j) {
11. cin >> c[i][j];
12. w[i][j] = j + 1;
13. }
14. }
15. cout << "\n";
16. vector< vector<int>> dp(n + 1, vector<int>(W + 1));
17. for (int j = 1; j <= W; ++j) {
18. for (int i = 1; i <= n; ++i) {
19. dp[i][j] = dp[i - 1][j];
20. for (int k = 1; k <= W; ++k) {
21. int wi = w[i - 1][k - 1];
22. int ci = c[i - 1][k - 1];
23. if(j - wi >= 0) {
24. dp[i][j] = max(dp[i][j], dp[i - 1][j - wi] + ci);
25. }
26. }
27. }
28. }
29. // for (int j = 0; j <= W; ++j) {
30. // for (int i = 0; i <= n; ++i) {
31. // cout << dp[i][j] << '\t';
32. // }
33. // cout << "\n";
34. // }
35. int ans = dp.back().back();
36. cout << ans << "\n";
37. }
38. int main () {
39. ios::sync\_with\_stdio(false);
40. cin.tie(0);
41. int t;
42. cin >> t;
43. for (int i = 0; i < t; ++i) {
44. solve();
45. }
46. return 0;
47. }

**Тесты**

****

**Задание 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.

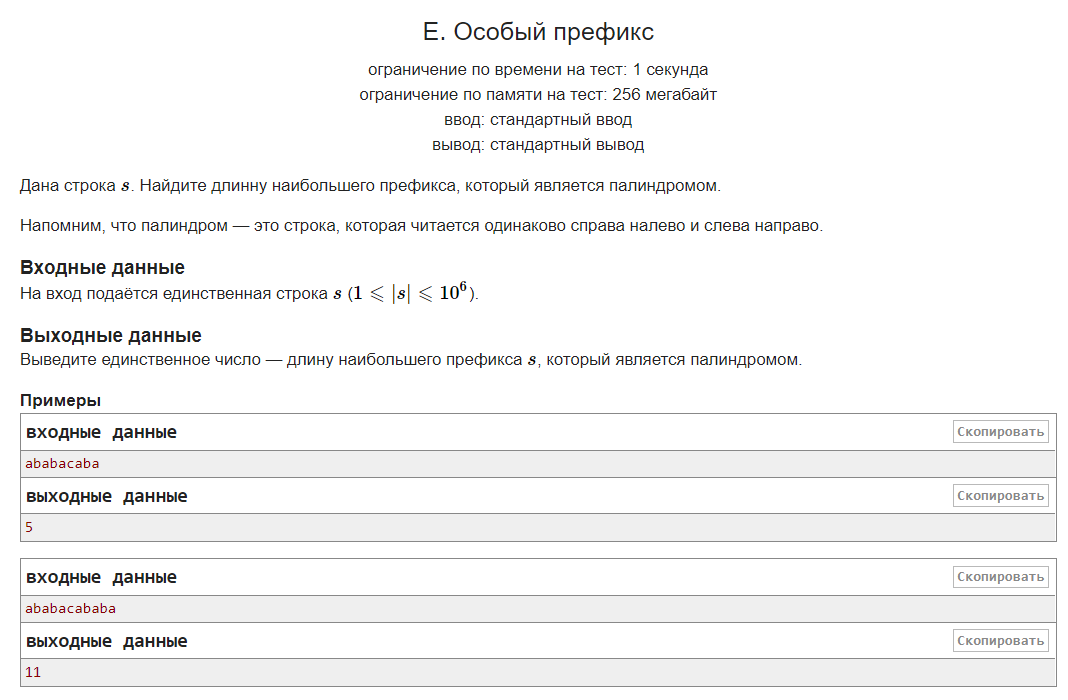
**Исходный код**

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**

****

**Задание 5.**

****

**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**

****