САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ

Факультет: Программной инженерии и компьютерной техники
Направление подготовки: 09.03.04 (Программная инженерия)

Дисциплина «Алгоритмы и структуры данных»

Отчёт по лабораторной работе Неделя №3

OpenEdu

Группа: P3217
Выполнил: Минин Александр

г. Санкт-Петербург
2018г.

Задача №1

Условие

В этой задаче Вам нужно будет отсортировать много неотрицательных целых чисел.

Вам даны два массива, A и B, содержащие соответственно n и m элементов. Числа, которые нужно будет отсортировать, имеют вид $A_i \cdot B_j$, где $1 \le i \le n$ и $1 \le j \le m$. Иными словами, каждый элемент первого массива нужно умножить на каждый элемент второго массива.

Пусть из этих чисел получится отсортированная последовательность C длиной $n \cdot m$. Выведите сумму каждого десятого элемента этой последовательности (то есть, $C_1 + C_{11} + C_{21} + \ldots$).

Формат входного файла

В первой строке содержатся числа n и m ($1 \le n, m \le 6000$) — размеры массивов. Во второй строке содержится n чисел — элементы массива A. Аналогично, в третьей строке содержится m чисел — элементы массива B. Элементы массива неотрицательны и не превосходят 40000.

Формат выходного файла

Выведите одно число — сумму каждого десятого элемента последовательности, полученной сортировкой попарных произведенй элементов массивов A и B.

Решение

```
package week3;

import mooc.EdxIO;

public class Week3_1RadixRecursive {

    private static final int DIG_S = 8;
    private static final int COUNT_SIZE = (int) Math.pow(2, DIG_S);
    private static final int INITIAL_MASK;
    private static final int MAX_EL = 40000 + 1;

    static {
        int mask = 0;
        for (int i = 0; i < DIG_S; i++) {
            mask |= 0x80000000 >> i;
        }
        INITIAL_MASK = mask;
    }

    private static EdxIO edxIO;

    private static int[] tmpResult;

    private static void solveBucket(final int[] C, final int digit, final int l, final int r) {
        if (digit == Integer.SIZE / DIG_S) {
            return;
        } else if (r - l < 10 && l % 10 != 0 && r % 10 > l % 10) {
            return;
        } else if (r - l < 47) {
            insertionSort(C, l, r);
        } else {
            countingSort(C, digit, l, r);
        }
```

```
}

private static void countingSort(final int[] C, final int digit, final int l, final int r) {
    int[] count = new int[COUNT_SIZE];

    int bitMask = INITIAL_MASK >>> (digit * DIG_S);
    int shift = (32 - (digit + 1) * DIG_S);

    int length = r - l;
    for (int j = l; j < r; j++) {
        count[(C[j] & bitMask) >> shift]++;
    }

    for (int k = 1; k < count.length; k++) {
        count[k] = count[k] + count[k - 1];
    }

    if (digit == Integer.SIZE / DIG_S - 1 && count[0] == length) {
        return;
    }

    int[] countClone = count.clone();

    for (int j = length - 1; j >= 0; j--) {
        int d = (C[l + j] & bitMask) >> shift;
        tmpResult[count[d] - 1] = C[l + j];
        count[d]--;
    }

    for (int i = 0; i < length; i++) {
        C[l + i] = tmpResult[i];
    }

    count = countClone;

    int left = l;
    if (count[0] > 0) {
        solveBucket(C, digit + 1, left, left + count[0]);
        left = l + count[0];
    }
    for (int k = 1; k < count.length; k++) {
        if (count[k] > count[k - 1]) {
            solveBucket(C, digit + 1, left, l + count[k]);
            left = l + count[k];
        }
        if (count[k] == length) {
            return;
        }
    }
}

private static void insertionSort(final int[] array, final int l, final int r) {
    for (int i = l; i < r; i++) {
        int j = i;
        while (j > l && array[j] < array[j - 1]) {
            int tmp = array[j];
            array[j] = array[j - 1];
            array[j - 1] = tmp;
            j--;
        }
```

```
    }
}

public static void main(String[] args) {
    solve();
}

private static void fillArrayFromCount(int[] arrayToFill, int[] count) {
    for (int i = 0, toIndex = 0; i < count.length; i++) {
        for (int k = 0; k < count[i]; k++) {
            arrayToFill[toIndex] = i;
            toIndex++;
        }
    }
}

private static void solve() {
    edxIO = EdxIO.create();

    int sizeA = edxIO.nextInt();
    int sizeB = edxIO.nextInt();

    int[] A = new int[sizeA];
    int[] B = new int[sizeB];
    int[] C = new int[sizeA * sizeB];

    int[] ACount = new int[MAX_EL];
    int[] BCount = new int[MAX_EL];

    for (int i = 0; i < sizeA; i++) {
        A[i] = edxIO.nextInt();
        ACount[A[i]]++;

    }

    for (int i = 0; i < sizeB; i++) {
        B[i] = edxIO.nextInt();
        BCount[B[i]]++;
    }

    fillArrayFromCount(A, ACount);
    fillArrayFromCount(B, BCount);

    if (B.length < A.length) {
        int[] tmp = A;
        A = B;
        B = tmp;
    }

    for (int i = 0, totalIndex = 0; i < A.length; i++) {
        for (int k = 0; k < B.length; k++, totalIndex++) {
            C[totalIndex] = A[i] * B[k];
        }
    }

    // init
    tmpResult = new int[C.length];

    solveBucket(C, 0, 0, C.length);
```

```
        long sum = 0;
        for (int i = 0; i < C.length; i += 10) {
            sum += C[i];
        }

        edxIO.println(sum);
        edxIO.close();
    }
}
```

Результаты

№ теста	Результат	Время (мс)	Память	Размер входного файла	Размер выходного файла
Max		1812	345944064	68699	18
1	OK	125	21958656	24	4
2	OK	125	21950464	34	3
3	OK	140	21917696	38	4
4	OK	125	22016000	106	12
5	OK	109	21962752	234	13
6	OK	125	22036480	698	13
7	OK	125	22032384	705	14
8	OK	125	22040576	586	14
9	OK	187	23642112	34325	14
10	OK	140	22265856	5769	14
11	OK	125	22102016	3498	14
12	OK	125	22155264	924	14
13	OK	125	22081536	3494	14
14	OK	125	22216704	5772	14
15	OK	140	23691264	34449	14
16	OK	156	24776704	34368	15
17	OK	140	24330240	4006	15
18	OK	125	24150016	2886	15
19	OK	171	24313856	4009	15
20	OK	140	24285184	34361	15
21	OK	250	39395328	34966	16
22	OK	234	38785024	9167	16
23	OK	281	40546304	9162	16
24	OK	281	39251968	34917	16

25	OK	687	103903232	39991	17
26	OK	546	106123264	28668	17
27	OK	515	104058880	40034	17
28	OK	1031	199688192	51489	17
29	OK	937	199925760	51525	17
30	OK	1703	344543232	68655	18
31	OK	1781	345927680	68625	18
32	OK	1812	345944064	68699	18

Задача №2

Условие

Дано n строк, выведите их порядок после k фаз цифровой сортировки.

Формат входного файла

В первой строке входного файла содержатся числа n — число строк, m — их длина и k – число фаз цифровой сортировки ($1 \le n \le 10^6$, $1 \le k \le m \le 10^6$, $n \cdot m \le 5 \cdot 10^7$). Далее находится описание строк, **но в нетривиальном формате**. Так, i-ая строка ($1 \le i \le n$) записана в i-ых символах второй, ..., $(m+1)$-ой строк входного файла. Иными словами, строки написаны по вертикали. **Это сделано специально, чтобы сортировка занимала меньше времени.**

Строки состоят из строчных латинских букв: от символа "a" до символа "z" включительно. В таблице символов ASCII все эти буквы располагаются подряд и в алфавитном порядке, код буквы "a" равен 97, код буквы "z" равен 122.

Формат выходного файла

Выведите номера строк в том порядке, в котором они будут после k фаз цифровой сортировки.

Решение

```
package week3;

import mooc.EdxIO;

import java.io.IOException;

public class Week3_2Fast {

    private static EdxIO edxIO;

    private static void countingSort(byte[] array, int l, int r, int[] mapping, boolean[] ranging) {
        int[] count = new int['z' - 'a' + 1];

        int length = r - l;
        for (int j = l; j < r; j++) {
            count[array[j] - 'a']++;
        }

        if (count[0] > 0) {
            ranging[l + count[0] - 1] = true;
        }
```

```
        for (int k = 1; k < count.length; k++) {
            count[k] = count[k] + count[k - 1];
            if (count[k] > count[k - 1]) {
                ranging[l + count[k] - 1] = true;
            }
        }

        int[] newMapping = new int[length];
        for (int j = length - 1; j >= 0; j--) {
            newMapping[count[array[l + j] - 'a'] - 1] = mapping[l + j];
            count[array[l + j] - 'a']--;
        }

        for (int i = 0; i < length; i++) {
            mapping[l + i] = newMapping[i];
        }
    }

    private static void propagate(byte[] array, int[] mapping) {
        byte[] tmp = array.clone();
        for (int i = 0; i < mapping.length; i++) {
            array[i] = tmp[mapping[i]];
        }
    }

    public static void main(String[] args) throws IOException {
        edxIO = EdxIO.create();

        int stringCount = edxIO.nextInt();
        int stringLength = edxIO.nextInt();
        int totalPhases = edxIO.nextInt();

        // skipping not required char sequences
        for (int i = 0; i < stringLength - totalPhases; i++) {
            edxIO.nextBytes();
        }

        // BEGIN initialization
        byte src[];
        boolean ranging[] = new boolean[stringCount];
        ranging[stringCount - 1] = true;

        int[] mapping = new int[stringCount];
        for (int i = 0; i < mapping.length; i++) {
            mapping[i] = i;
        }

        int minL = 0;
        // END initialization

        for (int i = 0; i < totalPhases; i++) {
            src = edxIO.nextBytes();
            propagate(src, mapping);

            int j = minL;
            while (j < stringCount && ranging[j]) {
                minL++;
                j++;
            }
            if (minL > stringCount - 1) {
```

```
                break;
            }

            int l = minL;
            while (j < stringCount) {
                if (ranging[j]) {
                    countingSort(src, l, j + 1, mapping, ranging);
                    l = j + 1;
                    j++;
                    // skipping sorted units
                    while (j < stringCount && ranging[j]) {
                        l++;
                        j++;
                    }
                } else {
                    j++;
                }
            }
        }

        // BEGING printing result
        for (int i = 0; i < mapping.length; i++) {
            edxIO.print(mapping[i] + 1 + " ");
        }
        // END printing result

        edxIO.close();
    }
}
```

Результаты

№ теста	Результат	Время, мс	Память	Размер входного файла	Размер выходного файла
Max		1312	211075072	52000020	6888896
1	OK	125	21630976	22	6
2	OK	140	21598208	22	6
3	OK	109	21602304	22	6
4	OK	187	21577728	10	2
5	OK	140	21606400	11	4
6	OK	140	21598208	130	21
7	OK	156	21594112	129	21
8	OK	125	21618688	129	21
9	OK	109	21549056	129	21
10	OK	125	21639168	129	21
11	OK	125	21557248	230	51
12	OK	109	21581824	229	51
13	OK	109	21635072	229	51

14	OK	156	21573632	229	51
15	OK	109	21598208	229	51
16	OK	125	21594112	450	51
17	OK	125	21557248	449	51
18	OK	140	21590016	450	51
19	OK	109	21614592	449	51
20	OK	125	21630976	449	51
21	OK	140	21557248	530	141
22	OK	109	21581824	529	141
23	OK	171	21614592	529	141
24	OK	125	21598208	529	141
25	OK	140	21594112	529	141
26	OK	109	21569536	1212	21
27	OK	109	21594112	1210	21
28	OK	125	21581824	1211	21
29	OK	109	21590016	1211	21
30	OK	125	21618688	1211	21
31	OK	156	21643264	2031	692
32	OK	109	21643264	2030	692
33	OK	140	21622784	2030	692
34	OK	125	21643264	2030	692
35	OK	140	21655552	2030	692
36	OK	156	21594112	2610	141
37	OK	140	21590016	2609	141
38	OK	109	21602304	2610	141
39	OK	93	21626880	2610	141
40	OK	109	21606400	2609	141
41	OK	109	21610496	4051	692
42	OK	125	21667840	4050	692
43	OK	140	21663744	4051	692
44	OK	125	21684224	4051	692
45	OK	109	21655552	4051	692
46	OK	140	21553152	6012	21
47	OK	140	21741568	6010	21
48	OK	156	21843968	6012	21

Дан массив из n элементов. Какие числа являются k_1-ым, (k_1+1)-ым, ..., k_2-ым в порядке неубывания в этом массиве?

Формат входного файла

В первой строке входного файла содержатся три числа: n — размер массива, а также границы интервала k_1 и k_2, при этом $2 \le n \le 4 \cdot 10^7, 1 \le k_1 \le k_2 \le n, k_2 - k_1 < 200$.

Во второй строке находятся числа A, B, C, a_1, a_2, по модулю не превосходящие 10^9. Вы должны получить элементы массива, начиная с третьего, по формуле: $a_i = A \cdot a_{i-2} + B \cdot a_{i-1} + C$. Все вычисления должны производится в 32-битном знаковом типе, переполнения должны игнорироваться.

Обращаем Ваше внимание, что массив из $4 \cdot 10^7$ 32-битных целых чисел занимает в памяти **160 мегабайт**! Будьте аккуратны!

Подсказка: эту задачу лучше всего решать модификацией быстрой сортировки. Однако сортировка массива целиком по времени, скорее всего, не пройдет, поэтому нужно подумать, как модифицировать быструю сортировку, чтобы не сортировать те части массива, которые не нужно сортировать.

> Эту задачу, скорее всего, **нельзя решить ни на Python, ни на PyPy**. Мы не нашли способа сгенерировать $4 \cdot 10^7$ 32-битных целых чисел и при этом уложиться в ограничение по времени. Если у Вас тоже не получается, попробуйте другой язык программирования, например, **Cython** (расширение файла `*.pyx`).

Формат выходного файла

В первой и единственной строке выходного файла выведите k_1-ое, (k_1+1)-ое, ..., k_2-ое в порядке неубывания числа в массиве a. Числа разделяйте одним пробелом.

49	OK	125	21798912	6012	21
50	OK	125	21749760	6010	21
51	OK	156	21635072	10213	292

52	OK	109	21721088	10211	292
53	OK	140	21815296	10212	292
54	OK	125	21766144	10212	292
55	OK	156	21721088	10212	292
56	OK	109	21839872	20052	3893
57	OK	140	21905408	20051	3893
58	OK	171	21950464	20052	3893
59	OK	171	22036480	20052	3893
60	OK	156	21954560	20051	3893
61	OK	125	21610496	26012	141
62	OK	125	21766144	26010	141
63	OK	125	22249472	26012	141
64	OK	109	21852160	26011	141
65	OK	140	22085632	26012	141
66	OK	109	21700608	40413	692
67	OK	125	21794816	40411	692
68	OK	156	22450176	40413	692
69	OK	125	21884928	40412	692
70	OK	171	21991424	40413	692
71	OK	93	21647360	52014	141
72	OK	109	21864448	52011	141
73	OK	125	23252992	52013	141
74	OK	171	22822912	52013	141
75	OK	140	23080960	52013	141
76	OK	109	21594112	102015	292
77	OK	156	22233088	102012	292
78	OK	171	24817664	102014	292
79	OK	187	24301568	102014	292
80	OK	109	23388160	102014	292
81	OK	203	29499392	200033	108894
82	OK	140	26353664	200032	108894
83	OK	171	30052352	200032	108894
84	OK	125	26697728	200032	108894
85	OK	156	28372992	200032	108894
86	OK	171	24317952	250112	23893

87	OK	156	23674880	250111	23893
88	OK	140	27357184	250112	23893
89	OK	140	24895488	250111	23893
90	OK	109	25985024	250112	23893
91	OK	203	28561408	400053	108894
92	OK	140	26611712	400052	108894
93	OK	218	30760960	400053	108894
94	OK	203	29802496	400053	108894
95	OK	218	30937088	400053	108894
96	OK	125	21958656	501014	3893
97	OK	125	23597056	501012	3893
98	OK	234	29982720	501014	3893
99	OK	156	25919488	501014	3893
100	OK	140	24264704	501013	3893
101	OK	203	24227840	1000414	23893
102	OK	156	25133056	1000412	23893
103	OK	343	33644544	1000414	23893
104	OK	203	29093888	1000413	23893
105	OK	265	32854016	1000414	23893
106	OK	109	21622784	2400018	21
107	OK	156	32100352	2400013	21
108	OK	359	59322368	2400018	21
109	OK	328	59301888	2400018	21
110	OK	296	59359232	2400018	21
111	OK	234	36990976	2500113	288894
112	OK	218	35794944	2500112	288894
113	OK	359	51679232	2500113	288894
114	OK	312	38567936	2500112	288894
115	OK	359	48599040	2500113	288894
116	OK	125	22175744	4004016	8893
117	OK	156	31100928	4004013	8893
118	OK	343	54276096	4004016	8893
119	OK	328	38862848	4004015	8893
120	OK	343	53166080	4004016	8893
121	OK	296	37928960	5000215	288894

122	OK	234	40898560	5000213	288894
123	OK	390	62693376	5000214	288894
124	OK	328	48001024	5000214	288894
125	OK	406	61972480	5000214	288894
126	OK	468	46833664	10000216	588895
127	OK	484	59408384	10000214	588895
128	OK	500	69046272	10000215	588895
129	OK	390	65499136	10000215	588895
130	OK	484	67883008	10000215	588895
131	OK	359	59461632	20000216	1288895
132	OK	359	77697024	20000214	1288895
133	OK	640	80732160	20000215	1288895
134	OK	484	79351808	20000215	1288895
135	OK	671	80154624	20000215	1288895
136	OK	265	37081088	25001015	288894
137	OK	250	80044032	25001013	288894
138	OK	625	84598784	25001015	288894
139	OK	500	82898944	25001015	288894
140	OK	515	83595264	25001015	288894
141	OK	140	21647360	26000018	141
142	OK	296	81051648	26000013	141
143	OK	750	116740096	26000018	141
144	OK	671	115830784	26000018	141
145	OK	531	89137152	26000018	141
146	OK	140	21762048	25100017	1892
147	OK	203	74067968	25100013	1892
148	OK	546	81969152	25100017	1892
149	OK	515	82751488	25100017	1892
150	OK	390	82178048	25100016	1892
151	OK	187	24285184	25010016	23893
152	OK	234	73437184	25010013	23893
153	OK	640	83132416	25010016	23893
154	OK	375	82272256	25010015	23893
155	OK	609	82362368	25010016	23893
156	OK	578	67252224	25000114	3388895

157	OK	468	86278144	25000113	3388895
158	OK	796	122908672	25000114	3388895
159	OK	625	90365952	25000114	3388895
160	OK	453	87232512	25000113	3388895
161	OK	171	22220800	40040018	8893
162	OK	218	94498816	40040014	8893
163	OK	890	102375424	40040018	8893
164	OK	609	97562624	40040017	8893
165	OK	750	97329152	40040018	8893
166	OK	109	21663744	40400019	692
167	OK	250	95686656	40400014	692
168	OK	906	102576128	40400019	692
169	OK	500	97697792	40400018	692
170	OK	406	97357824	40400016	692
171	OK	187	28971008	40004017	108894
172	OK	296	94838784	40004014	108894
173	OK	718	130658304	40004017	108894
174	OK	484	97648640	40004016	108894
175	OK	593	97566720	40004017	108894
176	OK	453	59269120	40000416	1288895
177	OK	437	98918400	40000414	1288895
178	OK	906	133947392	40000416	1288895
179	OK	609	99631104	40000415	1288895
180	OK	531	98238464	40000414	1288895
181	OK	140	21651456	51000019	292
182	OK	328	106110976	51000014	292
183	OK	1031	141897728	51000019	292
184	OK	515	108900352	51000018	292
185	OK	812	142135296	51000019	292
186	OK	171	21975040	50100018	3893
187	OK	250	104624128	50100014	3893
188	OK	906	140111872	50100018	3893
189	OK	546	107819008	50100018	3893
190	OK	734	107626496	50100018	3893
191	OK	703	106348544	50000115	6888896

192	OK	578	119853056	50000114	6888896
193	OK	1312	209477632	50000115	6888896
194	OK	1125	204472320	50000115	6888896
195	OK	1187	211075072	50000115	6888896
196	OK	109	21737472	50200019	1892
197	OK	250	104796160	50200014	1892
198	OK	890	139628544	50200018	1892
199	OK	734	107978752	50200018	1892
200	OK	812	108158976	50200018	1892
201	OK	390	46006272	50001016	588895
202	OK	453	108777472	50001014	588895
203	OK	984	142692352	50001016	588895
204	OK	718	140795904	50001016	588895
205	OK	593	108445696	50001015	588895
206	OK	328	36315136	50002017	288894
207	OK	343	105459712	50002014	288894
208	OK	906	142225408	50002016	288894
209	OK	734	134148096	50002016	288894
210	OK	984	140570624	50002016	288894
211	OK	531	69685248	50000216	3388895
212	OK	500	110977024	50000214	3388895
213	OK	1156	209555456	50000215	3388895
214	OK	1031	201269248	50000215	3388895
215	OK	843	148209664	50000215	3388895
216	OK	109	21635072	52000020	141
217	OK	296	107356160	52000014	141
218	OK	1140	207007744	52000019	141
219	OK	1046	142675968	52000019	141
220	OK	515	109527040	52000018	141
221	OK	156	26341376	50010017	48894
222	OK	296	104202240	50010014	48894
223	OK	812	140095488	50010017	48894
224	OK	468	106770432	50010017	48894
225	OK	468	106795008	50010017	48894
226	OK	171	24379392	50020018	23893

227	OK	234	104140800	50020014	23893
228	OK	984	140795904	50020017	23893
229	OK	828	112672768	50020017	23893
230	OK	890	112865280	50020017	23893