Университет ИТМО

Факультет программной инженерии и компьютерной техники

Лабораторная работа № 10

по дисциплине «Алгоритмы и структуры данных»

Openedu – неделя 10

Подготовил:

студент группы P3217

Бураков Илья Алексеевич

Преподаватели:

Романов Алексей Андреевич

Волчек Дмитрий Геннадьевич

Санкт-Петербург, 2019

Префикс-функция

Условие

Постройте префикс-функцию для всех непустых префиксов заданной строки s.

Формат входного файла

Первая строка входного файла содержит s $(1 \leq |s| \leq 10^6)$. Строка состоит из букв латинского алфавита.

Формат выходного файла

Выведите значения префикс-функции для всех префиксов строки s длиной $1, 2, \ldots, |s|$, в указанном порядке.

Примеры

input.txt	output.txt
aaaAAA	0 1 2 0 0 0
abacaba	0 0 1 0 1 2 3

Решение

openedu/week10/lab10_1.cpp

```
#include "edx-io.hpp"
#include <iostream>
#include <fstream>
#include <vector>
#include <algorithm>
#include <string>
#include <list>
#include <assert.h>

using namespace std;

int main() {
	string s;
	io >> s;

	auto p = vector<int>(s.size());

	int j = 0;
	for (int i = 1; i < s.size(); i++) {
		if (s[i] == s[j]) {
			p[i] = ++j;
		} else if (j == 0) {
			p[i] = 0;
		} else {
			j = p[j - 1];
			i--;
		}
	}

	for (auto e : p) io << e << " ";
	return 0;
}
```

Результаты

№ теста	Результат	Время, с	Память	Размер входного файла	Размер выходного файла
Max		0.234	52441088	1000002	6888889
1	OK	0.031	10465280	8	11
2	OK	0.015	10412032	9	13
3	OK	0.031	10592256	3	1
4	OK	0.031	10473472	4	3
5	OK	0.031	10493952	4	3
6	OK	0.031	10428416	12	19
7	OK	0.031	10448896	12	19
8	OK	0.062	14049280	92672	185339
9	OK	0.046	16429056	99998	588845
10	OK	0.046	16392192	100002	561028
11	OK	0.062	16351232	176391	352777
12	OK	0.062	21413888	199994	1288778
13	OK	0.062	21258240	199992	1190916
14	OK	0.062	16084992	172864	345723
15	OK	0.078	24875008	300002	1988889
16	OK	0.093	24166400	300002	1716598
17	OK	0.062	17178624	249367	498729
18	OK	0.109	28073984	400002	2688854
19	OK	0.093	27009024	399998	2333023
20	OK	0.078	22343680	455342	910679
21	OK	0.109	31211520	499996	3388797
22	OK	0.109	29626368	499998	2875815
23	OK	0.078	23011328	505139	1010273
24	OK	0.156	38027264	600000	4088810
25	OK	0.140	34594816	600002	3977734
26	OK	0.093	23699456	539096	1078187
27	OK	0.203	40128512	699998	4788806
28	OK	0.156	37699584	700002	4566499
29	OK	0.093	22794240	492073	984141
30	OK	0.218	41795584	799997	5488764
31	OK	0.156	35971072	800002	3984417
32	OK	0.125	26619904	763540	1527075
33	OK	0.187	44748800	899994	6188743
34	OK	0.187	42614784	900002	5662192
35	OK	0.125	27447296	836144	1672283
36	OK	0.218	52383744	1000002	6888854
37	OK	0.203	46895104	1000002	6555314
38	OK	0.187	29696000	1000002	1999999

39	OK	0.203	52428800	1000002	6888889
40	OK	0.203	52441088	1000002	6888889
41	OK	0.218	45699072	1000002	6209772
42	OK	0.234	52416512	1000002	6888884
43	OK	0.218	52412416	1000002	6888869
44	OK	0.203	49041408	1000002	6388894
45	OK	0.203	52346880	1000002	6883894

Z-функция

Условие

Постройте Z-функцию для заданной строки s.

Формат входного файла

Первая строка входного файла содержит s $(2 \leq |s| \leq 10^6)$. Строка состоит из букв латинского алфавита.

Формат выходного файла

Выведите значения Z-функции для всех индексов $2, 3, \ldots, |s|$ строки s, в указанном порядке.

Примеры

input.txt	output.txt
aaaAAA	2 1 0 0 0
abacaba	0 1 0 3 0 1

Решение

openedu/week10/lab10_2.cpp

```
#include "edx-io.hpp"
#include <iostream>
#include <fstream>
#include <vector>
#include <map>
#include <algorithm>
#include <string>
#include <list>
#include <assert.h>
#include <tuple>

using namespace std;
typedef unsigned long long ull;

int main() {
	string s;
	io >> s;

	vector<int> z(1000000);

	int l = 0, r = 0;

	for (int i = 1; i < s.size(); i++) {
		if (i >= r) {
```

```
                        // manually comparing
                        int j = 0;
                        while (i + j < s.size() && s[i + j] == s[j]) j++;
                        // updating z-block
                        l = i;
                        r = i + j;
                        // saving result
                        z[i] = j;
                } else {
                        // we're inside z-block
                        if (z[i - l] < r - i) {
                                // our entire known prefix fits inside z-block, so we've
already compared it
                                z[i] = z[i - l];
                        } else {
                                // known prefix outranges z-block, so we get more
symbols to manually compare
                                int j = r - i;
                                while (i + j < s.size() && s[i + j] == s[j]) j++;
                                l = i;
                                r = j + i;
                                z[i] = j;
                        }
                }
        }

        for (int i = 1; i < s.size(); i++) {
                io << z[i] << " ";
        }
        return 0;
}
```

Результаты

№ теста	Результат	Время, с	Память	Размер входного файла	Размер выходного файла
Max		0.250	7831552	1000002	6888888
1	OK	0.015	5824512	8	10
2	OK	0.000	5824512	9	12
3	OK	0.000	5824512	4	2
4	OK	0.000	5824512	4	2
5	OK	0.000	5824512	5	4
6	OK	0.015	5824512	12	18
7	OK	0.000	5816320	12	18
8	OK	0.031	6008832	92672	185338
9	OK	0.031	6025216	99998	264801
10	OK	0.015	6021120	100002	272211
11	OK	0.031	6180864	176391	352776
12	OK	0.046	6217728	199994	474050
13	OK	0.046	6221824	199992	456479
14	OK	0.046	6172672	172864	345722
15	OK	0.062	6426624	300002	1988888
16	OK	0.062	6426624	300002	786113
17	OK	0.046	6316032	249367	498728
18	OK	0.093	6623232	400002	1036108

19	OK	0.093	6623232	399998	885168
20	OK	0.093	6733824	455342	910678
21	OK	0.093	6828032	499996	1217153
22	OK	0.109	6828032	499998	1267986
23	OK	0.109	6836224	505139	1010272
24	OK	0.140	7024640	600000	1406340
25	OK	0.125	7024640	600002	1477776
26	OK	0.125	6897664	539096	1078186
27	OK	0.140	7221248	699998	1682393
28	OK	0.140	7221248	700002	1558331
29	OK	0.093	6811648	492073	984140
30	OK	0.171	7417856	799997	1804662
31	OK	0.156	7426048	800002	2196116
32	OK	0.156	7348224	763540	1527074
33	OK	0.187	7622656	899994	2030971
34	OK	0.203	7622656	900002	2765564
35	OK	0.171	7499776	836144	1672282
36	OK	0.234	7827456	1000002	2611108
37	OK	0.203	7827456	1000002	2227775
38	OK	0.203	7831552	1000002	1999998
39	OK	0.250	7831552	1000002	6888888
40	OK	0.234	7827456	1000002	6888888
41	OK	0.218	7831552	1000002	2841971
42	OK	0.250	7831552	1000002	4444443
43	OK	0.218	7831552	1000002	2977776
44	OK	0.218	7831552	1000002	2000043
45	OK	0.218	7831552	1000002	2004885

Декомпозиция строки

Условие

Строка `ABCABCDEDEDEF` содержит подстроку `ABC`, повторяющуюся два раза подряд, и подстроку `DE`, повторяющуюся три раза подряд. Таким образом, ее можно записать как `ABC*2+DE*3+F`, что занимает меньше места, чем исходная запись той же строки.

Ваша задача - построить наиболее экономное представление данной строки s в виде, продемонстрированном выше, а именно, подобрать такие $s_1, a_1, \ldots, s_k, a_k$, где s_i - строки, а a_i - числа, чтобы $s = s_1 \cdot a_1 + \ldots + s_k \cdot a_k$. Под операцией умножения строки на целое положительное число подразумевается конкатенация одной или нескольких копий строки, число которых равно числовому множителю, то есть, `ABC*2=ABCABC`. При этом требуется минимизировать общую длину итогового описания, в котором компоненты разделяются знаком `+`, а умножение строки на число записывается как умножаемая строка и множитель, разделенные знаком `*`. Если же множитель равен единице, его, вместе со знаком `*`, допускается не указывать.

Формат входного файла

Первая строка входного файла содержит s ($1 \le |s| \le 5 \cdot 10^3$). Строка состоит из букв латинского алфавита.

Формат выходного файла

Выведите оптимальное представление строки, данной во входном файле. Если оптимальных представлений несколько, выведите любое.

Примеры

input.txt	output.txt
ABCABCDEDEDEF	ABC*2+DE*3+F
Hello	Hello

Решение

openedu/week10/lab10_3.cpp

```
#include "edx-io.hpp"
#include <iostream>
#include <fstream>
#include <vector>
#include <map>
#include <algorithm>
#include <string>
#include <sstream>
#include <list>
#include <assert.h>
#include <tuple>

using namespace std;
typedef unsigned long long ull;

int group_repeats(string s, int cz) {
	// determine group length
	int l = 2;
	while (s.substr(l*cz, cz) == s.substr(0, cz)) l++;
	return l;
}

tuple<int, int> calc_z_func(string s, vector<int> &z) {
	z[0] = 0;
	int l = 0, r = 0;
	int maxz = 0;
	int maxl = 0;
	int cz = 0;
	int i;

	auto check_candidate = [&i, s, &maxz, &maxl](int candz) {
		if (candz == i && candz > maxz) {
			// got new candidate - new max z-prefix
			int l = group_repeats(s, candz);

			// HOW TO OPTIMIZE ???
			if (l >= maxl) {
				// new candidate is better
				maxz = candz;
				maxl = l;
			}
		}
	};
```

```
	auto grab_chars_while_can = [&i, &l, &r, &z, check_candidate, s](int j = 0)
{
		while (i + j < s.size() && s[i + j] == s[j]) {
			j++;
			check_candidate(j);
		}
		// updating z-block
		l = i;
		r = i + j;
		// saving result
		z[i] = j;
	};

	for (i = 1; i < s.size(); i++) {
		if (i >= r) {
			// manually comparing
			grab_chars_while_can();
		} else {
			// we're inside z-block
			if (z[i - l] < r - i) {
				// our entire known prefix fits inside z-block, so we've
already compared it
				z[i] = z[i - l];
			} else {
				// known prefix outranges z-block, so we get more
symbols to manually compare
				grab_chars_while_can(r - i);
			}
		}
		check_candidate(z[i]);
	}
	return make_tuple(maxz, maxl);
}

string get_group_mult_str(pair<string, int> g) {
	stringstream result;
	result << g.first;
	if (g.second != 1) {
		result << "*" << g.second;
	}
	return result.str();
}

string get_group_plain_str(pair<string, int> g) {
	stringstream result;
	while (g.second--) result << g.first;
	return result.str();
}

int main() {
	string s;
	io >> s;

	vector<int> z(s.size());
	vector<pair<string, int>> groups;


	int l = 0, r = 0;
	for (int i = 0; i < s.size();) {
		string ss = s.substr(i);
		int maxz, maxl;
		tie(maxz, maxl) = calc_z_func(ss, z);
		if (maxz) {
			groups.push_back(make_pair(ss.substr(0, maxz), maxl));
		} else {
			groups.push_back(make_pair(ss.substr(0, 1), 1));
```

```
                i++;
            }
            i += maxz * maxl;
        }

        // print groups
        bool plain = true;
        for (int i = 0; i < groups.size(); i++) {
            auto g = groups[i];

            if (!plain) {
                io << "+";
            }

            string mult = get_group_mult_str(g);
            if (mult.size() < g.first.size() * g.second) {
                if (i && plain) io << "+";
                io << mult;
                plain = false;
            } else {
                string plainstr = get_group_plain_str(g);
                io << plainstr;
                plain = true;
            }
        }
        return 0;
}
```

Результаты

№ теста	Результат	Время, с	Память	Размер входного файла	Размер выходного файла
Max		0.250	11956224	5002	5000
1	OK	0.031	10141696	15	12
2	OK	0.015	10125312	7	5
3	OK	0.015	10100736	3	1
4	OK	0.015	10289152	4	2
5	OK	0.046	10092544	5	3
6	OK	0.015	10141696	6	3
7	OK	0.031	10297344	7	3
8	OK	0.015	10121216	8	3
9	OK	0.031	10129408	9	3
10	OK	0.015	10096640	10	3
11	OK	0.031	10149888	11	3
12	OK	0.031	10137600	12	4
13	OK	0.031	10076160	13	4
14	OK	0.031	10088448	14	4
15	OK	0.015	10125312	4	2
16	OK	0.031	10067968	6	4
17	OK	0.031	10113024	8	4
18	OK	0.031	10096640	10	4
19	OK	0.031	10104832	12	4
20	OK	0.031	10039296	14	4

21	OK	0.031	10080256	16	4
22	OK	0.031	10096640	18	4
23	OK	0.015	10088448	20	4
24	OK	0.031	10137600	22	5
25	OK	0.031	10092544	24	5
26	OK	0.031	10080256	26	5
27	OK	0.031	10866688	450	448
28	OK	0.031	11264000	498	19
29	OK	0.031	11272192	498	56
30	OK	0.031	11186176	817	815
31	OK	0.046	11264000	1001	13
32	OK	0.046	11268096	1001	18
33	OK	0.046	11198464	1452	1450
34	OK	0.031	11255808	1502	6
35	OK	0.046	11284480	1502	73
36	OK	0.046	11243520	1828	1826
37	OK	0.062	11284480	1993	15
38	OK	0.062	11288576	1991	269
39	OK	0.046	11202560	1338	1336
40	OK	0.078	11288576	2499	15
41	OK	0.078	11300864	2486	188
42	OK	0.093	11243520	2402	2400
43	OK	0.109	11350016	3002	7
44	OK	0.109	11370496	2985	66
45	OK	0.078	11255808	2374	2372
46	OK	0.140	11313152	3502	14
47	OK	0.140	11296768	3500	10
48	OK	0.093	11268096	2227	2225
49	OK	0.156	11624448	4002	12
50	OK	0.171	11583488	4002	35
51	OK	0.109	11259904	2921	2919
52	OK	0.187	11763712	4502	22
53	OK	0.203	11362304	4501	86
54	OK	0.234	11800576	4765	4763
55	OK	0.234	11923456	5000	8
56	OK	0.218	11542528	4988	109
57	OK	0.250	11493376	5002	5000
58	OK	0.234	11894784	5002	6
59	OK	0.250	11956224	5002	31
60	OK	0.234	11513856	5002	82
61	OK	0.218	11915264	5002	7
62	OK	0.234	11952128	5002	10

63	OK	0.218	11943936	5002	503
64	OK	0.234	11956224	5002	54