Министерство образования и науки Российской Федерации

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

Факультет программной инженерии и компьютерной техники

Кафедра информатики и прикладной математики
Направление подготовки 09.03.04 Программная инженерия

Дисциплина «Алгоритмы и структуры данных»

ОТЧЁТ

по лабораторной работе №4
неделя четвертая

Выполнил:
Айгузин Иван Олегович
P3218

Преподаватели:

Романов Алексей Андреевич
Волчек Дмитрий Геннадьевич

Санкт-Петербург

2018

Стек

1.0 из 1.0 балла (оценивается)

Имя входного файла:	input.txt
Имя выходного файла:	output.txt
Ограничение по времени:	2 секунды
Ограничение по памяти:	256 мегабайт

Реализуйте работу стека. Для каждой операции изъятия элемента выведите ее результат.

На вход программе подаются строки, содержащие команды. Каждая строка содержит одну команду. Команда — это либо "+ N", либо "-". Команда "+ N" означает добавление в стек числа N, по модулю не превышающего 10^9. Команда "-" означает изъятие элемента из стека. Гарантируется, что не происходит извлечения из пустого стека. Гарантируется, что размер стека в процессе выполнения команд не превысит 10^6 элементов.

Формат входного файла

В первой строке входного файла содержится M ($1 \le M \le 10^6$) — число команд. Каждая последующая строка исходного файла содержит ровно одну команду.

Формат выходного файла

Выведите числа, которые удаляются из стека с помощью команды "-", по одному в каждой строке. Числа нужно выводить в том порядке, в котором они были извлечены из стека. Гарантируется, что изъятий из пустого стека не производится.

```
Using System;
using System.IO;
using System.Linq;

namespace Week04.Task01 {
    public sealed class Program {
        private static StreamReader In;
        private static StreamWriter Out;

        private static void Main(string[] args) {
            if (!args.Contains("console")) {
                SetupIO();
            }

            Run();

            if (args.Contains("console")) {
                Console.ReadLine();
            }

            DisposeIO();
        }

        private static void Run() {
            var n = int.Parse(Console.ReadLine());
```

```
            var stack = new string[1000000];
            var cursor = 0;

            for (var i = 0; i < n; i++) {
                var line = ReadLineArray();

                if (line[0] == "+") {
                    stack[cursor++] = line[1];
                    continue;
                }

                Console.WriteLine(stack[--cursor]);
            }
        }

        private static string[] ReadLineArray() {
            return Console.ReadLine()
                .Split(' ')
                .ToArray();
        }

        private static void SetupIO() {
            In = new StreamReader("input.txt");
            Out = new StreamWriter("output.txt");

            Console.SetIn(In);
            Console.SetOut(Out);
        }

        private static void DisposeIO() {
            In?.Dispose();
            Out?.Dispose();
        }
    }
}
```

№ теста	Результат	Время, с	Память	Размер входного файла	Размер выходного файла
Max		1.046	74186752	13389454	5693807
1	OK	0.031	10874880	33	10
2	OK	0.031	10903552	11	3
3	OK	0.031	10813440	19	6
4	OK	0.046	10788864	19	6
5	OK	0.031	10797056	19	6
6	OK	0.031	10809344	96	45
7	OK	0.031	10842112	85	56
8	OK	0.078	10809344	129	11
9	OK	0.031	10878976	131	12
10	OK	0.031	10862592	859	540
11	OK	0.031	10813440	828	573
12	OK	0.031	10858496	1340	11
13	OK	0.031	10883072	1325	12
14	OK	0.031	11083776	8292	5590
15	OK	0.031	11063296	8212	5706
16	OK	0.031	11153408	13298	111
17	OK	0.015	11169792	13354	12
18	OK	0.046	20221952	82372	56548
19	OK	0.031	20512768	82000	56993
20	OK	0.046	20840448	132796	1134
21	OK	0.015	20590592	133914	11
22	OK	0.078	20291584	819651	569557
23	OK	0.125	27865088	819689	569681
24	OK	0.125	29257728	1328670	11294
25	OK	0.109	29368320	1338543	11
26	OK	0.562	20361216	8196274	5693035
27	OK	0.812	50847744	8193816	5693807
28	OK	1.015	73146368	13286863	112020
29	OK	0.968	74186752	13389454	11
30	OK	1.046	74186752	13388564	11

Очередь

1.0 из 1.0 балла (оценивается)

Имя входного файла:	input.txt
Имя выходного файла:	output.txt
Ограничение по времени:	2 секунды
Ограничение по памяти:	256 мегабайт

Реализуйте работу очереди. Для каждой операции изъятия элемента выведите ее результат.

На вход программе подаются строки, содержащие команды. Каждая строка содержит одну команду. Команда — это либо «+ N», либо «-». Команда «+ N» означает добавление в очередь числа N, по модулю не превышающего 10^9. Команда «-» означает изъятие элемента из очереди. Гарантируется, что размер очереди в процессе выполнения команд не превысит 10^6 элементов.

Формат входного файла

В первой строке содержится M ($1 \le M \le 10^6$) — число команд. В последующих строках содержатся команды, по одной в каждой строке.

Формат выходного файла

Выведите числа, которые удаляются из очереди с помощью команды «-», по одному в каждой строке. Числа нужно выводить в том порядке, в котором они были извлечены из очереди. Гарантируется, что извлечения из пустой очереди не производится.

```
using System;
using System.IO;
using System.Linq;

namespace Week04.Task02 {
    public sealed class Program {
        private static StreamReader In;
        private static StreamWriter Out;

        private static void Main(string[] args) {
            if (!args.Contains("console")) {
                SetupIO();
            }

            Run();

            if (args.Contains("console")) {
                Console.ReadLine();
            }

            DisposeIO();
        }

        private static void Run() {
            var n = int.Parse(Console.ReadLine());
```

```
            var queue = new string[1000000];
            var cursorTop = 0;
            var cursorBottom = 0;

            for (var i = 0; i < n; i++) {
                var line = ReadLineArray();

                if (line[0] == "+") {
                    queue[cursorTop++] = line[1];
                    continue;
                }

                Console.WriteLine(queue[cursorBottom++]);
            }
        }

        private static string[] ReadLineArray() {
            return Console.ReadLine()
                .Split(' ')
                .ToArray();
        }

        private static void SetupIO() {
            In = new StreamReader("input.txt");
            Out = new StreamWriter("output.txt");

            Console.SetIn(In);
            Console.SetOut(Out);
        }

        private static void DisposeIO() {
            In?.Dispose();
            Out?.Dispose();
        }
    }
}
```

№ теста	Результат	Время, с	Память	Размер входного файла	Размер выходного файла
Max		1.046	74366976	13389454	5693807
1	OK	0.031	10829824	20	7
2	OK	0.031	10854400	11	3
3	OK	0.031	10825728	19	6
4	OK	0.015	10870784	19	6
5	OK	0.031	10854400	96	45
6	OK	0.031	10854400	85	56
7	OK	0.031	10788864	129	12
8	OK	0.031	10805248	131	12
9	OK	0.015	10842112	859	538
10	OK	0.031	10866688	828	573
11	OK	0.031	10895360	1340	12
12	OK	0.031	10862592	1325	12
13	OK	0.046	11104256	8292	5589
14	OK	0.031	11087872	8212	5706
15	OK	0.031	11182080	13298	115
16	OK	0.031	11182080	13354	12
17	OK	0.031	20316160	82372	56552
18	OK	0.062	20533248	82000	56993
19	OK	0.015	20774912	132796	1124
20	OK	0.031	20598784	133914	12
21	OK	0.093	26882048	819651	569553
22	OK	0.093	27865088	819689	569681
23	OK	0.125	29339648	1328670	11296
24	OK	0.109	29294592	1338543	12
25	OK	0.765	49885184	8196274	5693025
26	OK	0.796	50819072	8193816	5693807
27	OK	1.046	73674752	13286863	112110
28	OK	1.015	74366976	13389454	10
29	OK	1.000	74149888	13388564	11

Скобочная последовательность

1.0 из 1.0 балла (оценивается)

Имя входного файла:	input.txt
Имя выходного файла:	output.txt
Ограничение по времени:	2 секунды
Ограничение по памяти:	256 мегабайт

Последовательность A, состоящую из символов из множества «(», «)», «[» и «]», назовем *правильной скобочной последовательностью*, если выполняется одно из следующих утверждений:

- A — пустая последовательность;
- первый символ последовательности A — это «(», и в этой последовательности существует такой символ «)», что последовательность можно представить как $A = (B)C$, где B и C — правильные скобочные последовательности;
- первый символ последовательности A — это «[», и в этой последовательности существует такой символ «]», что последовательность можно представить как $A = [B]C$, где B и C — правильные скобочные последовательности.

Так, например, последовательности «(())» и «()[]» являются правильными скобочными последовательностями, а последовательности «[)» и «((» таковыми не являются.

Входной файл содержит несколько строк, каждая из которых содержит последовательность символов «(», «)», «[» и «]». Для каждой из этих строк выясните, является ли она правильной скобочной последовательностью.

Формат входного файла

Первая строка входного файла содержит число N ($1 \le N \le 500$) - число скобочных последовательностей, которые необходимо проверить. Каждая из следующих N строк содержит скобочную последовательность длиной от 1 до 10^4 включительно. В каждой из последовательностей присутствуют только скобки указанных выше видов.

Формат выходного файла

Для каждой строки входного файла выведите в выходной файл «YES», если соответствующая последовательность является правильной скобочной последовательностью, или «NO», если не является.

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;

namespace Week04.Task03 {
    public sealed class Program {
        private static StreamReader _in;
        private static StreamWriter _out;

        private static void Main(string[] args) {
            if (!args.Contains("console")) {
                SetupIO();
```

```
        }

        Run();

        if (args.Contains("console")) {
            Console.ReadLine();
        }

        DisposeIO();
    }

    private static void Run() {
        var n = int.Parse(Console.ReadLine());
        var stack = new Stack<char>();

        var braces = new Dictionary<char, char> {
            ['('] = ')',
            ['['] = ']'
        };

        for (var i = 0; i < n; i++) {
            stack.Clear();
            CheckLine(Console.ReadLine());
        }

        void CheckLine(string line) {
            foreach (var c in line) {
                if (braces.ContainsKey(c)) {
                    stack.Push(c);
                    continue;
                }

                if (stack.Count == 0) {
                    Console.WriteLine("NO");
                    return;
                }

                if (braces[stack.Pop()] == c) {
                    continue;
                }

                Console.WriteLine("NO");
                return;
            }

            Console.WriteLine(stack.Count == 0 ? "YES" : "NO");
        }
    }
```

```
        private static string[] ReadLineArray() {
            return Console.ReadLine()
                .Split(' ')
                .ToArray();
        }

        private static void SetupIO() {
            _in = new StreamReader("input.txt");
            _out = new StreamWriter("output.txt");

            Console.SetIn(_in);
            Console.SetOut(_out);
        }

        private static void DisposeIO() {
            _in?.Dispose();
            _out?.Dispose();
        }
    }
}
```

№ теста	Результат	Время, с	Память	Размер входного файла	Размер выходного файла
Max		0.281	13672448	5000885	2133
1	OK	0.046	11599872	31	22
2	OK	0.031	11595776	15	16
3	OK	0.031	11620352	68	66
4	OK	0.046	11624448	324	256
5	OK	0.046	11644928	1541	1032
6	OK	0.031	11685888	5880	2128
7	OK	0.046	11780096	50867	2129
8	OK	0.062	12886016	500879	2110
9	OK	0.281	13656064	5000884	2120
10	OK	0.250	13672448	5000885	2133

Очередь с минимумом

2.0 из 2.0 баллов (оценивается)

Имя входного файла:	input.txt
Имя выходного файла:	output.txt
Ограничение по времени:	2 секунды
Ограничение по памяти:	256 мегабайт

Реализуйте работу очереди. В дополнение к стандартным операциям очереди, необходимо также отвечать на запрос о минимальном элементе из тех, которые сейчас находится в очереди. Для каждой операции запроса минимального элемента выведите ее результат.

На вход программе подаются строки, содержащие команды. Каждая строка содержит одну команду. Команда — это либо «+ N», либо «-», либо «?». Команда «+ N» означает добавление в очередь числа N, по модулю не превышающего 10^9. Команда «-» означает изъятие элемента из очереди. Команда «?» означает запрос на поиск минимального элемента в очереди.

Формат входного файла

В первой строке содержится M ($1 \leq M \leq 10^6$) — число команд. В последующих строках содержатся команды, по одной в каждой строке.

Формат выходного файла

Для каждой операции поиска минимума в очереди выведите её результат. Результаты должны быть выведены в том порядке, в котором эти операции встречаются во входном файле. Гарантируется, что операций извлечения или поиска минимума для пустой очереди не производится.

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;

namespace Week04.Task04 {
    public sealed class Program {
        private static StreamReader _in;
        private static StreamWriter _out;

        private static void Main(string[] args) {
            if (!args.Contains("console")) {
                SetupIO();
            }

            Run();

            if (args.Contains("console")) {
                Console.ReadLine();
            }

            DisposeIO();
        }
```

```
        private static void Run() {
            var n = int.Parse(Console.ReadLine());
            var queue = new Queue<int>();

            var mins = new LinkedList<int>();

            for (var i = 0; i < n; i++) {
                var line = ReadLineArray();

                switch (line[0]) {
                    case "+":
                        var a = int.Parse(line[1]);
                        queue.Enqueue(a);

                        while (mins.Count > 0 && mins.First.Value > a) {
                            mins.RemoveFirst();
                        }

                        mins.AddFirst(a);
                        break;
                    case "-":
                        var b = queue.Dequeue();

                        if (mins.Last.Value == b) {
                            mins.RemoveLast();
                        }

                        break;
                    default:
                        Console.WriteLine(mins.Last.Value);
                        break;
                }
            }
        }

        private static string[] ReadLineArray() {
            return Console.ReadLine()
                .Split(' ')
                .ToArray();
        }

        private static void SetupIO() {
            _in = new StreamReader("input.txt");
            _out = new StreamWriter("output.txt");

            Console.SetIn(_in);
            Console.SetOut(_out);
        }
```

```
        private static void DisposeIO() {
            _in?.Dispose();
            _out?.Dispose();
        }
    }
}
```

№ теста	Результат	Время, с	Память	Размер входного файла	Размер выходного файла
Max		1.265	81973248	13389342	4002151
1	OK	0.031	11481088	29	10
2	OK	0.031	11472896	11	3
3	OK	0.031	11542528	22	6
4	OK	0.031	11497472	22	6
5	OK	0.031	11513856	36	9
6	OK	0.031	11489280	48	12
7	OK	0.015	11485184	76	35
8	OK	0.031	11513856	129	12
9	OK	0.093	11456512	67	48
10	OK	0.031	11501568	44	9
11	OK	0.046	11440128	45	9
12	OK	0.015	11468800	44	9
13	OK	0.031	11489280	45	9
14	OK	0.031	11534336	721	384
15	OK	0.031	11505664	1340	12
16	OK	0.046	11522048	640	407
17	OK	0.031	11501568	445	90
18	OK	0.046	11587584	456	100
19	OK	0.031	11481088	445	90
20	OK	0.031	11505664	456	100
21	OK	0.031	11685888	6616	3812
22	OK	0.031	11890688	13389	12
23	OK	0.031	11710464	6461	4008
24	OK	0.031	11706368	4896	1140
25	OK	0.031	11694080	5007	1250
26	OK	0.031	11718656	4896	1140
27	OK	0.031	11739136	5007	1250
28	OK	0.078	12705792	64907	39589
29	OK	0.046	12718080	133814	12
30	OK	0.031	12685312	64675	39996
31	OK	0.046	13299712	53897	13890
32	OK	0.031	12673024	55008	15000
33	OK	0.031	13340672	53897	13890
34	OK	0.046	12623872	55008	15000
35	OK	0.093	12783616	645271	404305
36	OK	0.109	13791232	1338956	12
37	OK	0.093	13328384	646300	400008
38	OK	0.109	21024768	588898	163890
39	OK	0.078	13430784	600009	175000

40	OK	0.125	21032960	588898	163890
41	OK	0.093	13402112	600009	175000
42	OK	0.640	12771328	6465010	4002151
43	OK	0.796	20992000	13389342	12
44	OK	0.625	16113664	6462989	4000004
45	OK	0.859	54722560	6388899	1888890
46	OK	0.609	16740352	6500010	2000000
47	OK	0.890	54702080	6388899	1888890
48	OK	0.609	16736256	6500010	2000000
49	OK	0.781	20963328	13388086	12
50	OK	0.031	11472896	55	16
51	OK	0.031	11513856	705	225
52	OK	0.031	11747328	6506	2000
53	OK	0.046	13402112	65007	20000
54	OK	0.109	19877888	650008	200000
55	OK	0.828	42688512	6675213	2000000
56	OK	0.015	11489280	117	12
57	OK	0.031	11550720	1327	12
58	OK	0.046	11853824	13417	12
59	OK	0.046	14499840	133845	12
60	OK	0.140	22982656	1339319	12
61	OK	1.265	81973248	13388955	12

Quack

2.0 из 2.0 баллов (оценивается)

Имя входного файла:	input.txt
Имя выходного файла:	output.txt
Ограничение по времени:	2 секунды
Ограничение по памяти:	256 мегабайт

Язык Quack — забавный язык, который фигурирует в одной из задач с Internet Problem Solving Contest. В этой задаче вам требуется написать интерпретатор языка Quack.

Виртуальная машина, на которой исполняется программа на языке Quack, имеет внутри себя очередь, содержащую целые числа по модулю 65536 (то есть, числа от 0 до 65535, соответствующие беззнаковому 16-битному целому типу). Слово get в описании операций означает извлечение из очереди, put — добавление в очередь. Кроме того, у виртуальной машины есть 26 регистров, которые обозначаются буквами от 'a' до 'z'. Изначально все регистры хранят нулевое значение. В языке Quack существуют следующие команды (далее под α и β подразумеваются некие абстрактные временные переменные):

+	Сложение: get α, get β, put $(\alpha + \beta) \bmod 65536$
-	Вычитание: get α, get β, put $(\alpha - \beta) \bmod 65536$
*	Умножение: get α, get β, put $(\alpha \cdot \beta) \bmod 65536$
/	Целочисленное деление: get α, get β, put α div β (будем считать, что α div 0 = 0)
%	Взятие по модулю: get α, get β, put α mod β (будем считать, что α mod 0 = 0)
>[register]	Положить в регистр: get α, установить значение [register] в α
<[register]	Взять из регистра: put значение [register]
P	Напечатать: get α, вывести α в стандартный поток вывода и перевести строку
P[register]	Вывести значение регистра [register] в стандартный поток вывода и перевести строку
C	Вывести как символ: get α, вывести символ с ASCII-кодом α mod 256 в стандартный поток вывода
C[register]	Вывести регистр как символ: вывести символ с ASCII-кодом α mod 256 (где α — значение регистра [register]) в стандартный поток вывода
:[label]	Метка: эта строка программы имеет метку [label]
J[label]	Переход на строку с меткой [label]
Z[register] [label]	Переход если 0: если значение регистра [register] равно нулю, выполнение программы продолжается с метки [label]
E[register1] [register2] [label]	Переход если равны: если значения регистров [register1] и [register2] равны, исполнение программы продолжается с метки [label]
G[register1] [register2] [label]	Переход если больше: если значение регистра [register1] больше, чем значение регистра [register2], исполнение программы продолжается с метки [label]
Q	Завершить работу программы. Работа также завершается, если выполнение доходит до конца программы
[number]	Просто число во входном файле — put это число

Формат входного файла

Входной файл содержит синтаксически корректную программу на языке Quack. Известно, что программа завершает работу не более чем за 10^5 шагов. Программа содержит не менее одной и не более 10^5 инструкций. **Метки имеют длину от 1 до 10 и состоят из цифр и латинских букв.**

Формат выходного файла

Выведите содержимое стандартного потока вывода виртуальной машины в выходной файл.

```
Using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;

namespace Week04.Task05 {
    public sealed class Program {
        private static StreamWriter _out;

        private static void Main(string[] args) {
            if (!args.Contains("console")) {
                SetupIO();
            }

            new Quack().Run(File.ReadAllLines("input.txt"));

            if (args.Contains("console")) {
                Console.ReadLine();
            }

            DisposeIO();
        }

        private static void SetupIO() {
            _out = new StreamWriter("output.txt");
            Console.SetOut(_out);
        }

        private static void DisposeIO() {
            _out?.Dispose();
        }
    }

    internal class Quack {
        private readonly Queue<ushort> _queue = new Queue<ushort>();

        private Dictionary<int, ushort> _registers;

        private int _cursor;

        private Dictionary<char, Action<string>> _instructions;
```

```
    private Dictionary<string, int> _labels;
    private bool _stopped;

    public void Run(string[] lines) {
        _registers = Enumerable.Range('a', 26)
            .ToDictionary(x => x, y => (ushort) 0);

        _queue.Clear();
        DefineLabels(lines);
        DefineInstructions();

        while (_cursor < lines.Length && !_stopped) {
            Interpret(lines[_cursor]);
            _cursor += 1;
        }
    }

    private void Interpret(string s) {
        if (char.IsDigit(s[0])) {
            _queue.Enqueue(ushort.Parse(s));
            return;
        }

        _instructions[s[0]](s);
    }

    private void DefineInstructions() {
        _instructions = new Dictionary<char, Action<string>> {
            ['+'] = s => { Put(Get() + Get()); },
            ['-'] = s => { Put(Get() - Get()); },
            ['*'] = s => { Put(Get() * Get()); },
            ['/'] = s => {
                var a = Get();
                var b = Get();
                Put(b == 0 ? 0 : a / b);
            },
            ['%'] = s => {
                var a = Get();
                var b = Get();
                Put(b == 0 ? 0 : a % b);
            },
            ['>'] = s => { _registers[s[1]] = Get(); },
            ['<'] = s => { Put(_registers[s[1]]); },
            ['P'] = s => { Console.WriteLine(s.Length == 1 ? Get() : _registers[s[1]]); },
            ['C'] = s => { Console.Write((char) ((s.Length == 1 ? Get() : _registers[s[1]]) % 256)); },
            [':'] = s => { },
            ['J'] = s => { _cursor = _labels[new string(s.Skip(1).ToArray())]; },
            ['Z'] = s => {
```

```
                if (_registers[s[1]] == 0) {
                    _cursor = _labels[new string(s.Skip(2).ToArray())];
                }
            },
            ['E'] = s => {
                if (_registers[s[1]] == _registers[s[2]]) {
                    _cursor = _labels[new string(s.Skip(3).ToArray())];
                }
            },
            ['G'] = s => {
                if (_registers[s[1]] > _registers[s[2]]) {
                    _cursor = _labels[new string(s.Skip(3).ToArray())];
                }
            },
            ['Q'] = s => { _stopped = true; }
        };

        ushort Get() {
            return _queue.Dequeue();
        }

        void Put(int value) {
            _queue.Enqueue((ushort) (value % 65536));
        }
    }

    private void DefineLabels(string[] lines) {
        _labels = lines
            .Select((s, i) => (label: s, line: i))
            .Where(s => s.label[0] == ':')
            .ToDictionary(tuple => new string(tuple.label.Skip(1).ToArray()), tuple => tuple.line);
    }
  }
}
```

№ теста	Результат	Время, с	Память	Размер входного файла	Размер выходного файла
Max		0.187	31924224	1349803	250850
1	OK	0.078	13193216	69	6
2	OK	0.046	13201408	232	218
3	OK	0.046	12898304	3	0
4	OK	0.046	12972032	100	19
5	OK	0.078	14409728	56	58890
6	OK	0.062	14295040	67	30000
7	OK	0.062	14286848	67	30000
8	OK	0.078	14327808	55	30000
9	OK	0.046	12914688	461	60
10	OK	0.046	13217792	11235	21
11	OK	0.046	13369344	23748	42
12	OK	0.062	14094336	66906	8905
13	OK	0.046	13012992	7332	954
14	OK	0.046	13033472	4611	602
15	OK	0.062	13590528	37968	5424
16	OK	0.046	12955648	14	2
17	OK	0.046	12922880	70	10
18	OK	0.046	12910592	350	50
19	OK	0.046	12988416	1750	250
20	OK	0.062	13074432	8750	1250
21	OK	0.062	13631488	43750	6250
22	OK	0.062	16863232	218750	31250
23	OK	0.046	13508608	34606	4721
24	OK	0.078	20467712	683180	7
25	OK	0.093	20447232	683102	0
26	OK	0.187	31924224	1349803	0
27	OK	0.093	21082112	491572	247791
28	OK	0.093	21098496	491488	249618
29	OK	0.093	21090304	491600	249600
30	OK	0.093	21110784	491502	250850
31	OK	0.078	21106688	491416	249477
32	OK	0.078	21078016	491520	250262
33	OK	0.093	21086208	491317	246859
34	OK	0.093	21078016	491514	248199
35	OK	0.093	21102592	491557	249601