Санкт-Петербургский национальный исследовательский университет
информационных технологий, механики и оптики

Кафедра информатики и прикладной математики

Лабораторная работа №8
Дисциплина «Алгоритмы и структуры данных»

Выполнил:
Молодецкий Арсений Алексеевич
Группа P3217

Санкт-Петербург
2019

Задание №1

Реализуйте множество с операциями «добавление ключа», «удаление ключа», «проверка существования ключа».

Формат входного файла

В первой строке входного файла находится строго положительное целое число операций N, не превышающее $5 \cdot 10^5$. В каждой из последующих N строк находится одна из следующих операций:

- `A` x — добавить элемент x в множество. Если элемент уже есть в множестве, то ничего делать не надо.
- `D` x — удалить элемент x. Если элемента x нет, то ничего делать не надо.
- `?` x — если ключ x есть в множестве, выведите `Y`, если нет, то выведите `N`.

Аргументы указанных выше операций — целые числа, не превышающие по модулю 10^{18}.

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

namespace EightWeek
{
    class Program
    {
        private static StreamWriter _out;

        private static void Main(string[] args)
        {
            _out = new StreamWriter("output.txt");
            Console.SetOut(_out);
            new SetRunner().Run(File.ReadAllLines("input.txt"));
            DisposeIO();
        }

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

    }

    public class SetRunner
    {
        private Set _set;
        private Dictionary<char, Action<long>> instructions;

        public SetRunner()
        {
            _set = new Set();
            this.InitializeCommands();
        }

        public void InitializeCommands()
        {
            instructions = new Dictionary<char, Action<long>>
            {
                ['A'] = (number) => { _set.Add(number); },
                ['D'] = (number) => { _set.Delete(number); },
```

```
                ['?'] = (number) => { Console.WriteLine(_set.Find(number) ?
'Y' : 'N'); }
            };
        }
        public void Run(string[] commands)
        {
            for (int i = 1; i < commands.Length; i++)
            {
                var args = commands[i].Split();
                var number = long.Parse(args[1]);
                instructions[args[0][0]](number);
            }
        }
    }

    public class Set
    {
        private LinkedList<long>[] _array;
        private int _divider;

        public Set(int primeNumber = 65537)
        {
            _divider = primeNumber;
            _array = new LinkedList<long>[_divider * 2 - 1];
        }

        private long Hash(long number)
        {
            return number % _divider + 65536;
        }

        private bool isListExists(long number)
        {
            return _array[Hash(number)] == null ? false : true;
        }

        public void Add(long number)
        {
            if (!isListExists(number))
            {
                _array[Hash(number)] = new LinkedList<long>();
                _array[Hash(number)].AddFirst(number);
            }
            else
            {
                if (!Find(number,true))
                {
                    _array[Hash(number)].AddFirst(number);
                }
            }

        }

        public bool Find(long number, bool isListChecked = false)
        {
            if (isListChecked || isListExists(number))
            {
                return _array[Hash(number)].Find(number) == null ? false :
true;
            }
```

```
            return false;
        }

        public void Delete(long number)
        {
            if (Find(number))
            {
                _array[Hash(number)].Remove(number);
            }
        }
    }
}
```

№ теста	Результат	Время, с	Память	Размер входного файла	Размер выходного файла
Max		1.640	87478272	11189636	501237
1	OK	0.031	11476992	43	9
2	OK	0.031	11268096	8	3
3	OK	0.015	11427840	51	12
4	OK	0.031	11440128	542	99
5	OK	0.031	11444224	618	54
6	OK	0.031	11735040	5451	1038
7	OK	0.015	11661312	6436	957
8	OK	0.046	12759040	13382	957
9	OK	0.031	12902400	22394	981
10	OK	0.062	11694080	7030	465
11	OK	0.031	11747328	7020	411
12	OK	0.031	13471744	63829	10002
13	OK	0.031	14602240	80339	4947
14	OK	0.031	14696448	80203	5034
15	OK	0.109	25407488	545113	100323
16	OK	0.109	25837568	639485	99282
17	OK	0.125	26038272	738870	99558
18	OK	0.156	35373056	1338668	99636
19	OK	0.203	31612928	2237627	99540
20	OK	0.140	36438016	903052	50202
21	OK	0.140	36528128	902843	49536
22	OK	0.312	49213440	2725205	501237
23	OK	0.312	54505472	3196877	499713
24	OK	0.343	54472704	3694712	501051
25	OK	0.796	63426560	6694340	500355
26	OK	0.890	75169792	11189636	500040
27	OK	0.765	70516736	4902931	249012
28	OK	0.765	70479872	4902757	250305
29	OK	1.125	79446016	9687139	300000
30	OK	1.093	79429632	9687570	300000
31	OK	1.093	77254656	8000008	300000
32	OK	1.640	87478272	11000008	150000

Задание №2

Реализуйте прошитый ассоциативный массив.

Формат входного файла

В первой строке входного файла находится строго положительное целое число операций N, не превышающее $5 \cdot 10^5$. В каждой из последующих N строк находится одна из следующих операций:

- `get` x — если ключ x есть в множестве, выведите соответствующее ему значение, если нет, то выведите `<none>`.
- `prev` x — вывести значение, соответствующее ключу, находящемуся в ассоциативном массиве, который был вставлен позже всех, но до x, или `<none>`, если такого нет или в массиве нет x.
- `next` x — вывести значение, соответствующее ключу, находящемуся в ассоциативном массиве, который был вставлен раньше всех, но после x, или `<none>`, если такого нет или в массиве нет x.
- `put` x y — поставить в соответствие ключу x значение y. При этом следует учесть, что:
  - если, независимо от предыстории, этого ключа на момент вставки в массиве не было, то он считается только что вставленным и оказывается самым последним среди добавленных элементов — то есть, вызов `next` с этим же ключом сразу после выполнения текущей операции `put` должен вернуть `<none>`;
  - если этот ключ уже есть в массиве, то значение необходимо изменить, и в этом случае ключ не считается вставленным еще раз, то есть, не меняет своего положения в порядке добавленных элементов.
- `delete` x — удалить ключ x. Если ключа x в ассоциативном массиве нет, то ничего делать не надо.

Ключи и значения — строки из латинских букв длиной не менее одного и не более 20 символов.

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

namespace ItmoAlgos
{
    public class Program
    {

        private static string[] _input;
        private static int _currentLineIndex;

        private static string ReadLine()
        {
            return _input[_currentLineIndex++];
        }

        public static void Main(string[] args)
        {
            var dictionary = new Dictionary<string,
LinkedListNode<string>>();
            var linkedList = new LinkedList<string>();
            var output = new StringBuilder();
            var emptyNode = new LinkedListNode<string>("<none>");
            LinkedListNode<string> node;

            _input = File.ReadAllLines("input.txt");

            long n = long.Parse(ReadLine());
```

```
            for (long i = 0; i < n; i++)
            {
                string[] strings = ReadLine().Split();
                string key = strings[1];
                string value = strings.Length == 3 ? strings[2] : null;

                switch (strings[0])
                {
                    case "put":
                        if (dictionary.TryGetValue(key, out node))
                        {
                            node.Value = value;
                        }
                        else
                        {
                            dictionary[key] = linkedList.AddLast(value);
                        }
                        break;

                    case "get":
                        output.AppendLine((dictionary.GetValueOrDefault(key)
?? emptyNode).Value);
                        break;

                    case "prev":
                        output.AppendLine((dictionary[key].Previous ??
emptyNode).Value);
                        break;

                    case "next":
                        output.AppendLine((dictionary[key].Next ??
emptyNode).Value);
                        break;

                    case "delete":
                        if (dictionary.Remove(key, out node))
                        {
                            linkedList.Remove(node);
                        }
                        break;
                }
            }

            File.WriteAllText("output.txt", output.ToString());
        }
    }
}
```

№ теста	Результат	Время, с	Память	Размер входного файла	Размер выходного файла
Max		1.484	231497728	23499808	10303658
1	OK	0.031	10711040	158	26
2	OK	0.031	10661888	12	8
3	OK	0.031	10706944	25	5
4	OK	0.015	10686464	25	8
5	OK	0.031	10735616	82	20
6	OK	0.015	10764288	1200	504
7	OK	0.031	10809344	1562	564
8	OK	0.031	11034624	12204	4617
9	OK	0.015	11063296	12058	4340
10	OK	0.109	27574272	960183	395964
11	OK	0.109	27574272	1318345	765350
12	OK	0.109	27582464	1420595	880052
13	OK	0.093	27566080	1079934	395020
14	OK	0.093	27631616	840022	332970
15	OK	0.093	27590656	1223121	889998
16	OK	0.171	32641024	3120970	486100
17	OK	0.156	32575488	3123298	486652
18	OK	0.156	32628736	3122193	479024
19	OK	0.093	27566080	900630	420456
20	OK	0.156	32677888	3121195	486718
21	OK	0.250	55635968	4199992	8
22	OK	0.250	54345728	4099993	8
23	OK	0.281	53624832	3999994	8
24	OK	0.265	53669888	3899995	8
25	OK	0.234	52146176	3799996	8
26	OK	0.250	51306496	3699997	8
27	OK	0.250	50814976	3599998	8
28	OK	0.234	50794496	3499999	8
29	OK	0.250	49586176	3400000	8
30	OK	0.250	52854784	3300001	8
31	OK	0.328	52699136	5399043	1973124
32	OK	0.296	52756480	4200443	1669405
33	OK	0.328	52699136	6099290	4429770
34	OK	0.671	96837632	15598672	2589784
35	OK	0.687	96694272	15589269	2586758
36	OK	0.640	97554432	15603830	2398360
37	OK	0.281	52723712	4499616	2110630
38	OK	0.687	96878592	15603381	2583188
39	OK	1.390	212172800	20999992	8
40	OK	1.375	204955648	20499993	8
41	OK	1.375	206127104	19999994	8
42	OK	1.359	206753792	19499995	8
43	OK	1.343	202342400	18999996	8
44	OK	1.328	199991296	18499997	8
45	OK	1.343	201744384	17999998	8
46	OK	1.296	201728000	17499999	8
47	OK	1.328	196354048	17000000	8
48	OK	1.281	193589248	16500001	8
49	OK	1.281	146960384	18500008	5499986
50	OK	1.453	231497728	23499808	220
51	OK	0.484	72683520	13500208	10303658
52	OK	0.812	100020224	15500008	8799944
53	OK	1.359	204316672	21500008	2200000
54	OK	1.140	147173376	18500008	5500000
55	OK	1.484	231071744	23499808	220
56	OK	0.468	75546624	13500208	10300130
57	OK	0.828	100052992	15500008	8799958
58	OK	1.359	204382208	21500008	2200000

Задание №3

В данной задаче у Вас не будет проблем ни с вводом, ни с выводом. Просто реализуйте быструю хеш-таблицу.

В этой хеш-таблице будут храниться целые числа из диапазона $[0; 10^{15} - 1]$. Требуется поддерживать добавление числа x и проверку того, есть ли в таблице число x. Числа, с которыми будет работать таблица, генерируются следующим образом. Пусть имеется четыре целых числа N, X, A, B, такие что:

- $1 \le N \le 10^7$;
- $0 \le X < 10^{15}$;
- $0 \le A < 10^3$;
- $0 \le B < 10^{15}$.

Требуется N раз выполнить следующую последовательность операций:

- Если X содержится в таблице, то установить $A \leftarrow (A + A_C) \bmod 10^3$, $B \leftarrow (B + B_C) \bmod 10^{15}$.
- Если X не содержится в таблице, то добавить X в таблицу и установить $A \leftarrow (A + A_D) \bmod 10^3$, $B \leftarrow (B + B_D) \bmod 10^{15}$.
- Установить $X \leftarrow (X \cdot A + B) \bmod 10^{15}$.

Начальные значения X, A и B, а также N, A_C, B_C, A_D и B_D даны во входном файле. Выведите значения X, A и B после окончания работы.

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

namespace ADS.Week8
{
    public class HashTable
    {
        private int tableSize;
        private long[] table;

        public HashTable(int size)
        {
            tableSize = size;
            table = new long[size];
            for (int i = 0; i < size; i++)
                table[i] = -1;
        }

        public bool TryInsert(long key)
        {
            int hash = GetHash(key);
            int hash2 = GetHash2(key);
            while (table[hash] != -1 && table[hash] != key)
            {
                hash = (hash + hash2) % tableSize;
                hash2++;
            }
            if (table[hash] == key)
                return false;
            table[hash] = key;
```

```
            return true;
        }

        private int GetHash(long key)
        {
            return Math.Abs(unchecked((int)((long)(key * 47))) ^ (int)((key *
31) >> 32)) % tableSize;
        }
        private int GetHash2(long key)
        {
            return Math.Abs(unchecked((int)((long)(key * 113))) ^ (int)((key
* 97) >> 32)) % (tableSize - 1) + 1;
        }
    }
    public class Task3
    {
        public static void Main(string[] args)
        {
            using (StreamReader streamReader = new StreamReader("input.txt"))
            using (StreamWriter streamWriter = new
StreamWriter("output.txt"))
            {
                string[] str = streamReader.ReadLine().Split(' ');
                int n = int.Parse(str[0]);
                long x = long.Parse(str[1]);
                int a = int.Parse(str[2]);
                long b = long.Parse(str[3]);
                str = streamReader.ReadLine().Split(' ');
                int ac = int.Parse(str[0]);
                long bc = long.Parse(str[1]);
                int ad = int.Parse(str[2]);
                long bd = long.Parse(str[3]);

                HashTable hashTable = new HashTable(n * 2);
                for (int i = 0; i < n; i++)
                {
                    if (hashTable.TryInsert(x))
                    {
                        a = (a + ad) % 1000;
                        b = (b + bd) % 1000000000000000;
                    }
                    else
                    {
                        a = (a + ac) % 1000;
                        b = (b + bc) % 1000000000000000;
                    }
                    x = (x * a + b) % 1000000000000000;
                }
                streamWriter.WriteLine("{0} {1} {2}", x, a, b);
            }
        }
    }
}
```

№ теста	Результат	Время, с	Память	Размер входного файла	Размер выходного файла
Max		2.062	170541056	87	37
1	OK	0.031	10178560	18	7
2	OK	0.015	10178560	19	7
3	OK	0.031	10199040	21	7
4	OK	0.031	10186752	21	7
5	OK	0.046	10178560	21	7
6	OK	0.015	10186752	21	15
7	OK	0.031	10203136	21	7
8	OK	0.031	10235904	21	9
9	OK	0.031	10227712	21	9
10	OK	0.031	10252288	30	28
11	OK	0.031	10182656	30	28
12	OK	0.015	10211328	35	35
13	OK	0.015	10194944	47	32
14	OK	0.015	10190848	63	35
15	OK	0.031	10170368	81	37
16	OK	0.031	10203136	82	37
17	OK	0.031	11771904	23	7
18	OK	0.015	11792384	23	7
19	OK	0.031	11751424	23	7
20	OK	0.046	11816960	23	21
21	OK	0.031	11833344	23	7
22	OK	0.031	11767808	23	9
23	OK	0.031	11816960	23	9
24	OK	0.046	11804672	32	30
25	OK	0.031	11771904	32	30
26	OK	0.031	11771904	37	35
27	OK	0.031	11796480	51	35
28	OK	0.031	11780096	64	34
29	OK	0.062	11792384	84	37
30	OK	0.046	11792384	84	36
31	OK	0.062	26230784	24	7
32	OK	0.078	26218496	24	7
33	OK	0.062	26238976	24	7
34	OK	0.171	26189824	24	24
35	OK	0.062	26218496	24	7
36	OK	0.078	26284032	24	9
37	OK	0.078	26165248	24	9
38	OK	0.171	26198016	33	16
39	OK	0.140	26189824	33	30
40	OK	0.187	26185728	38	35
41	OK	0.171	26157056	52	35
42	OK	0.171	26181632	66	35
43	OK	0.187	26247168	84	37
44	OK	0.187	26193920	85	37
45	OK	0.500	170500096	25	7
46	OK	0.531	170491904	25	7
47	OK	0.546	170467328	25	7
48	OK	2.062	170463232	25	27
49	OK	0.500	170405888	25	7
50	OK	0.500	170496000	25	9
51	OK	0.500	170467328	25	9
52	OK	1.562	170430464	34	16
53	OK	1.343	170463232	34	30
54	OK	2.062	170455040	39	35
55	OK	2.000	170455040	51	35
56	OK	2.046	170528768	67	35
57	OK	2.015	170487808	87	37
58	OK	2.015	170532864	87	37
59	OK	2.015	170508288	87	35
60	OK	2.015	170459136	86	37
61	OK	2.031	170450944	87	37
62	OK	2.015	170442752	86	37
63	OK	2.015	170463232	86	37
64	OK	2.000	170430464	86	37
65	OK	2.015	170455040	87	37
66	OK	2.031	170463232	85	35
67	OK	2.015	170430464	85	36
68	OK	2.015	170541056	87	36