Санкт-Петербургский Национальный Исследовательский
Университет Информационных Технологий, Механики и Оптики
Мегафакультет компьютерных технологий и управления

Дисциплина
«Алгоритмы и структуры данных»

Лабораторная работа №8

Выполнил:
Студент группы P3218
Рябов Сергей Витальевич
Преподаватель:
Муромцев Дмитрий Ильич

Санкт-Петербург,
2018

1. Множество

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}.

Формат выходного файла

Выведите последовательно результат выполнения всех операций «?». Следуйте формату выходного файла из примера.

Исходный код (C#):

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

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

        public HashTable(int size = 1500)
        {
            tableSize = size;
            table = new LinkedList<long>[size];
            for (int i = 0; i < size; i++)
                table[i] = new LinkedList<long>();
        }

        public void Insert(long key)
        {
            int hash = GetHash(key);
            if (!table[hash].Contains(key))
                table[hash].AddLast(key);
        }
        public void Remove(long key)
        {
            int hash = GetHash(key);
            table[hash].Remove(key);
        }

        public bool ContainsKey(long key)
        {
            int hash = GetHash(key);
            return table[hash].Contains(key);
        }
```

```
        private int GetHash(long key)
        {
            return Math.Abs((((int)key).GetHashCode() + (key >> 32).GetHashCode()) % tableSize);
        }
    }
    public class Task1
    {
        public static void Main(string[] args)
        {
            using (StreamReader streamReader = new StreamReader("input.txt"))
            using (StreamWriter streamWriter = new StreamWriter("output.txt"))
            {
                int n = int.Parse(streamReader.ReadLine());
                HashTable hashTable = new HashTable(n);
                for (int i = 0; i < n; i++)
                {
                    string[] str = streamReader.ReadLine().Split(' ');
                    switch(str[0][0])
                    {
                        case 'A':
                            hashTable.Insert(long.Parse(str[1]));
                            break;
                        case 'D':
                            hashTable.Remove(long.Parse(str[1]));
                            break;
                        case '?':
                            streamWriter.WriteLine(hashTable.ContainsKey(long.Parse(str[1])) ? "Y" : "N");
                            break;
                    }
                }
            }
        }
    }
}
```

Результат:

№ теста	Результат	Время, с	Память	Размер входного файла	Размер выходного файла
Max		1.812	69259264	11189636	501237
1	OK	0.031	11026432	43	9
2	OK	0.015	11018240	8	3
3	OK	0.046	11010048	51	12
4	OK	0.031	11079680	542	99
5	OK	0.031	11087872	618	54
6	OK	0.031	11288576	5451	1038
7	OK	0.031	11292672	6436	957
8	OK	0.031	11358208	13382	957
9	OK	0.046	11452416	22394	981
10	OK	0.046	11325440	7030	465
11	OK	0.031	11292672	7020	411
12	OK	0.031	14151680	63829	10002
13	OK	0.046	14409728	80339	4947

2. Прошитый ассоциативный массив

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 putдолжен 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.

Формат выходного файла

Выведите последовательно результат выполнения всех операций get, prev, next. Следуйте формату выходного файла из примера.

Исходный код (C#):

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

namespace ADS.Week8
{
    public class HashTable
    {
        Dictionary<string, LinkedListNode<string>> table;
        private LinkedList<string> history;

        public HashTable()
        {
            table = new Dictionary<string, LinkedListNode<string>>();
            history = new LinkedList<string>();
        }

        public void Insert(string key, string value)
```

```
    {
        if (table.ContainsKey(key))
        {
            table[key].Value = value;
        }
        else
        {
            history.AddLast(value);
            table.Add(key, history.Last);
        }
    }
    public bool TryGetValue(string key, out string value)
    {
        if (table.TryGetValue(key, out LinkedListNode<string> node))
        {
            value = node.Value;
            return true;
        }
        value = null;
        return false;
    }
    public bool TryGetNext(string key, out string value)
    {
        if (table.TryGetValue(key, out LinkedListNode<string> node) && node.Next != null)
        {
            value = table[key].Next.Value;
            return true;
        }
        value = null;
        return false;
    }
    public bool TryGetPrevious(string key, out string value)
    {
        if (table.TryGetValue(key, out LinkedListNode<string> node) && node.Previous != null)
        {
            value = table[key].Previous.Value;
            return true;
        }
        value = null;
        return false;
    }
    public void Remove(string key)
    {
        if (table.TryGetValue(key, out LinkedListNode<string> node))
        {
            table.Remove(key);
            history.Remove(node);
        }
    }
}
public class Task2
{
    public static void Main(string[] args)
    {
        using (StreamReader streamReader = new StreamReader("input.txt"))
        using (StreamWriter streamWriter = new StreamWriter("output.txt"))
        {
            int n = int.Parse(streamReader.ReadLine());
            HashTable hashTable = new HashTable();
            for (int i = 0; i < n; i++)
            {
                string[] str = streamReader.ReadLine().Split(' ');
                switch(str[0])
                {
```

```
                        case "get":
                            streamWriter.WriteLine(hashTable.TryGetValue(str[1], out string value1) ?
value1 : "<none>");
                            break;
                        case "prev":
                            streamWriter.WriteLine(hashTable.TryGetPrevious(str[1], out string value2) ?
value2 : "<none>");
                            break;
                        case "next":
                            streamWriter.WriteLine(hashTable.TryGetNext(str[1], out string value3) ?
value3 : "<none>");
                            break;
                        case "put":
                            hashTable.Insert(str[1], str[2]);
                            break;
                        case "delete":
                            hashTable.Remove(str[1]);
                            break;
                    }
                }
            }
        }
    }
}
```

Результат:

№ теста	Результат	Время, с	Память	Размер входного файла	Размер выходного файла
Max		1.281	169656320	23499808	10303658
1	OK	0.046	10866688	158	26
2	OK	0.031	10735616	12	8
3	OK	0.046	10821632	25	5
4	OK	0.031	10817536	25	8
5	OK	0.046	11067392	82	20
6	OK	0.031	11022336	1200	504
7	OK	0.031	10924032	1562	564
8	OK	0.031	11182080	12204	4617
9	OK	0.031	11186176	12058	4340
10	OK	0.062	12099584	960183	395964
11	OK	0.078	12091392	1318345	765350
12	OK	0.078	12058624	1420595	880052
13	OK	0.078	12066816	1079934	395020
14	OK	0.062	12029952	840022	332970
15	OK	0.078	12009472	1223121	889998
16	OK	0.109	21925888	3120970	486100
17	OK	0.125	21966848	3123298	486652
18	OK	0.109	21999616	3122193	479024
19	OK	0.062	12066816	900630	420456
20	OK	0.125	21938176	3121195	486718

3. Почти интерактивная хеш-таблица

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 $A \leftarrow (A+AC) \bmod 10^3$, $B \leftarrow (B+BC) \bmod 10^{15}$.
- Если X не содержится в таблице, то добавить X в таблицу и установить $A \leftarrow (A+AD) \bmod 10^3$, $B \leftarrow (B+BD) \bmod 10^{15}$.
- Установить $X \leftarrow (X \cdot A+B) \bmod 10^{15}$.

Начальные значения X, A и B, а также N, AC, BC, AD и BD даны во входном файле. Выведите значения X, A и B после окончания работы.

Формат входного файла

Во первой строке входного файла содержится четыре целых числа N, X, A, B. Во второй строке содержится еще четыре целых числа AC, BC, AD и BD, такие что $0 \le AC,AD < 10^3$, $0 \le BC,BD < 10^{15}$.

Формат выходного файла

Выведите значения X, A и B после окончания работы.

Исходный код (C#):

```
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);
            }
        }
    }
}
```

Результат:

№ теста	Результат	Время, с	Память	Размер входного файла	Размер выходного файла
Max		2.062	170520576	87	37
1	OK	0.031	10174464	18	7
2	OK	0.015	10194944	19	7
3	OK	0.031	10264576	21	7
4	OK	0.031	10194944	21	7
5	OK	0.015	10244096	21	7
6	OK	0.031	10186752	21	15
7	OK	0.031	10248192	21	7
8	OK	0.031	10162176	21	9
9	OK	0.031	10280960	21	9
10	OK	0.015	10203136	30	28
11	OK	0.015	10244096	30	28
12	OK	0.031	10186752	35	35
13	OK	0.031	10194944	47	32
14	OK	0.015	10182656	63	35
15	OK	0.046	10244096	81	37
16	OK	0.015	10248192	82	37
17	OK	0.031	11784192	23	7
18	OK	0.031	11841536	23	7
19	OK	0.031	11780096	23	7
20	OK	0.046	11829248	23	21