Университет ИТМО

Факультет программной инженерии и компьютерной техники

Лабораторная работа № 8

по дисциплине «Алгоритмы и структуры данных»

Openedu – неделя 8

Подготовил:

студент группы P3217

Бураков Илья Алексеевич

Преподаватели:

Романов Алексей Андреевич

Волчек Дмитрий Геннадьевич

Санкт-Петербург, 2019

Множество

Условие

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.

Пример

input.txt	output.txt
8 A 2 A 5 A 3 ? 2 ? 4 A 2 D 2 ? 2	Y N N

Примечание

Эту задачу можно решить совершенно разными способами, включая использование различных средств стандартных библиотек (правда, не всех - в стандартных библиотеках некоторых языков программирования используются слишком предсказуемые методы хеширования). Именно по этой причине ее разумно использовать для проверки реализаций хеш-таблиц, которые понадобятся в следующих задачах этой недели. После окончания текущей порции экспериментов, пожалуйста, не забудьте сдать правильное решение, чтобы эта задача была зачтена!

Решение

openedu/week8/lab8_1.cpp

```
#include "edx-io.hpp"
#include <vector>
#include <algorithm>
#include <forward_list>

using namespace std;

#define HASHTABLE_SIZE 909151

unsigned int h(long long x) {
	x = ((x >> 16) ^ x) * 0x45d9f3b;
	x = ((x >> 16) ^ x) * 0x45d9f3b;
```

```
	x = (x >> 16) ^ x;
	return x % HASHTABLE_SIZE;
}

int main() {
	int n;
	io >> n;

	auto hashmap = new forward_list<long long>[HASHTABLE_SIZE];

	for (int i = 0; i < n; i++) {
		char c;
		long long arg;
		io >> c >> arg;
		switch (c) {
			case 'A': {
				hashmap[h(arg)].remove(arg);
				hashmap[h(arg)].push_front(arg);
			} break;
			case 'D': {
				hashmap[h(arg)].remove(arg);
			} break;
			case '?': {
				auto list = hashmap[h(arg)];
				io << (find(list.begin(), list.end(), arg) != list.end()
? "Y" : "N") << "\n";
			} break;
		}
	}

	return 0;
}
```

Результаты

№ теста	Результат	Время, с	Память	Размер входного файла	Размер выходного файла
Max		0.187	34676736	11189636	501237
1	OK	0.015	9506816	43	9
2	OK	0.015	9506816	8	3
3	OK	0.015	9506816	51	12
4	OK	0.015	9510912	542	99
5	OK	0.015	9519104	618	54
6	OK	0.015	9502720	5451	1038
7	OK	0.015	9523200	6436	957
8	OK	0.000	9519104	13382	957
9	OK	0.015	9523200	22394	981
10	OK	0.015	9531392	7030	465
11	OK	0.015	9527296	7020	411
12	OK	0.015	9510912	63829	10002
13	OK	0.015	9682944	80339	4947
14	OK	0.015	9674752	80203	5034
15	OK	0.031	9633792	545113	100323
16	OK	0.031	9736192	639485	99282
17	OK	0.031	9895936	738870	99558

18	OK	0.031	11546624	1338668	99636
19	OK	0.046	12451840	2237627	99540
20	OK	0.031	11673600	903052	50202
21	OK	0.031	11673600	902843	49536
22	OK	0.093	11812864	2725205	501237
23	OK	0.093	12292096	3196877	499713
24	OK	0.109	12865536	3694712	501051
25	OK	0.125	21237760	6694340	500355
26	OK	0.140	25731072	11189636	500040
27	OK	0.171	22106112	4902931	249012
28	OK	0.171	22102016	4902757	250305
29	OK	0.171	31494144	9687139	300000
30	OK	0.171	31490048	9687570	300000
31	OK	0.156	29806592	8000008	300000
32	OK	0.187	34676736	11000008	150000

Прошитый ассоциативный массив

Условие

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.

Формат выходного файла

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

Пример

input.txt	output.txt
14	c
put zero a	b
put one b	d
put two c	c
put three d	a
put four e	e
get two	<none>
prev two	
next two	
delete one	
delete three	
get two	
prev two	
next two	
next four	

Решение

openedu/week8/lab8_2.cpp

```
#include "edx-io.hpp"
#include <vector>
#include <algorithm>
#include <string>
#include <list>

using namespace std;

#define HASHTABLE_SIZE 800000

struct Entry {
	string key;
	string val;
	list<Entry*>::iterator orderlink;

	Entry(string newkey, string newval) {
		key = newkey;
		val = newval;
	}
};

void main() {
	int n;
	io >> n;

	auto hashmap = new list<Entry>[HASHTABLE_SIZE];
	auto orderlist = list<Entry*>();

	for (int i = 0; i < n; i++) {
		string cmd, key, val;
		io >> cmd >> key;

		auto &list = hashmap[hash<string>{}(key) % HASHTABLE_SIZE];
```

```
            auto result = find_if(list.begin(), list.end(), [key](const Entry& e)
{return e.key == key; });

            if (cmd == "put") {
                  io >> val;

                  if (result == list.end()) {
                        list.push_front(Entry(key, val));
                        orderlist.push_front(&list.front());
                        list.front().orderlink = orderlist.begin();
                  } else {
                        result->val = val;
                  }
            } else if (cmd == "delete") {
                  if (result != list.end()) {
                        orderlist.erase(result->orderlink);
                        list.erase(result);
                  }
            } else if (cmd == "get") {
                  io << (result != list.end() ? result->val : "<none>") << "\n";
            } else if (cmd == "next") {
                  io << ((result != list.end() && result->orderlink !=
orderlist.begin()) ? (*prev(result->orderlink))->val : "<none>") << "\n";
            } else if (cmd == "prev") {
                  io << ((result != list.end() && next(result->orderlink) !=
orderlist.end()) ? (*next(result->orderlink))->val : "<none>") << "\n";
            }
      }
}
```

Результаты

№ теста	Результат	Время, с	Память	Размер входного файла	Размер выходного файла
Max		0.859	227897344	23499808	10303658
1	OK	0.109	92213248	158	26
2	OK	0.109	92213248	12	8
3	OK	0.109	92213248	25	5
4	OK	0.093	92209152	25	8
5	OK	0.109	92213248	82	20
6	OK	0.109	92209152	1200	504
7	OK	0.109	92225536	1562	564
8	OK	0.109	92213248	12204	4617
9	OK	0.093	92217344	12058	4340
10	OK	0.125	92749824	960183	395964
11	OK	0.140	93106176	1318345	765350
12	OK	0.140	93216768	1420595	880052
13	OK	0.140	92868608	1079934	395020
14	OK	0.125	92639232	840022	332970
15	OK	0.140	93020160	1223121	889998
16	OK	0.187	99450880	3120970	486100
17	OK	0.203	99459072	3123298	486652
18	OK	0.203	99524608	3122193	479024

19	OK	0.125	92696576	900630	420456
20	OK	0.187	99450880	3121195	486718
21	OK	0.218	113725440	4199992	8
22	OK	0.218	113623040	4099993	8
23	OK	0.218	113524736	3999994	8
24	OK	0.218	113426432	3899995	8
25	OK	0.218	113324032	3799996	8
26	OK	0.203	113225728	3699997	8
27	OK	0.218	113127424	3599998	8
28	OK	0.203	113029120	3499999	8
29	OK	0.218	112930816	3400000	8
30	OK	0.203	112824320	3300001	8
31	OK	0.234	97189888	5399043	1973124
32	OK	0.234	95993856	4200443	1669405
33	OK	0.296	97898496	6099290	4429770
34	OK	0.546	128020480	15598672	2589784
35	OK	0.531	127737856	15589269	2586758
36	OK	0.531	130023424	15603830	2398360
37	OK	0.234	96292864	4499616	2110630
38	OK	0.546	128028672	15603381	2583188
39	OK	0.687	201281536	20999992	8
40	OK	0.671	200781824	20499993	8
41	OK	0.687	200282112	19999994	8
42	OK	0.687	199778304	19499995	8
43	OK	0.703	199278592	18999996	8
44	OK	0.703	198778880	18499997	8
45	OK	0.687	198279168	17999998	8
46	OK	0.671	197779456	17499999	8
47	OK	0.687	197279744	17000000	8
48	OK	0.687	196780032	16500001	8
49	OK	0.859	166625280	18500008	5499986
50	OK	0.843	227897344	23499808	220
51	OK	0.453	105299968	13500208	10303658
52	OK	0.828	129847296	15500008	8799944
53	OK	0.843	203411456	21500008	2200000
54	OK	0.828	166625280	18500008	5500000
55	OK	0.859	227897344	23499808	220
56	OK	0.500	105295872	13500208	10300130
57	OK	0.859	129847296	15500008	8799958
58	OK	0.843	203407360	21500008	2200000

Почти интерактивная хеш-таблица

Условие

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.

Формат входного файла

Во первой строке входного файла содержится четыре целых числа N, X, A, B. Во второй строке содержится еще четыре целых числа A_C, B_C, A_D и B_D, такие что $0 \le A_C, A_D < 10^3$, $0 \le B_C, B_D < 10^{15}$.

Формат выходного файла

Выведите значения X, A и B после окончания работы.

Пример

input.txt	output.txt
4 0 0 0 1 1 0 0	3 1 1

Решение

openedu/week8/lab8_3.cpp

```
#include "edx-io.hpp"
#include <vector>
#include <algorithm>
#include <string>
#include <forward_list>

using namespace std;
```

```
#define HASHTABLE_SIZE 800000

typedef unsigned long long ull;

auto HASHMAP = new vector<ull>[HASHTABLE_SIZE];

ull h(ull x) {
	x = ((x >> 16) ^ x) * 0x45d9f3b;
	x = ((x >> 16) ^ x) * 0x45d9f3b;
	x = (x >> 16) ^ x;
	return x % HASHTABLE_SIZE;
}

bool add_if_not_in_hashmap(ull x) {
	auto& list = HASHMAP[h(x)];
	if (find(list.begin(), list.end(), x) != list.end()) {
		// already found
		return false;
	} else {
		list.push_back(x);
		return true;
	}
}

#define E3 1000
#define E15 1000000000000000

void main() {
	long long N, X, A, B, Ac, Bc, Ad, Bd;
	io >> N >> X >> A >> B >> Ac >> Bc >> Ad >> Bd;


	for (int i = 0; i < N; i++) {
		if (add_if_not_in_hashmap(X)) {
			A = (A + Ad) % E3;
			B = (B + Bd) % E15;
		} else {
			A = (A + Ac) % E3;
			B = (B + Bc) % E15;
		}

		X = (X * A + B) % E15;
	}
	io << X << " " << A << " " << B << "\n";
}
```

Результаты

№ теста	Результат	Время, с	Память	Размер входного файла	Размер выходного файла
Max		4.000	176107520	87	37
1	OK	0.015	21438464	18	7
2	OK	0.031	21438464	19	7
3	OK	0.046	21434368	21	7
4	OK	0.015	21438464	21	7
5	OK	0.015	21434368	21	7
6	OK	0.015	21454848	21	15
7	OK	0.015	21442560	21	7
8	OK	0.015	21446656	21	9

9	OK	0.031	21442560	21	9
10	OK	0.031	21463040	30	28
11	OK	0.015	21463040	30	28
12	OK	0.015	21454848	35	35
13	OK	0.015	21458944	47	32
14	OK	0.015	21454848	63	35
15	OK	0.015	21458944	81	37
16	OK	0.015	21463040	82	37
17	OK	0.015	21434368	23	7
18	OK	0.015	21434368	23	7
19	OK	0.015	21434368	23	7
20	OK	0.031	23117824	23	21
21	OK	0.015	21442560	23	7
22	OK	0.015	21446656	23	9
23	OK	0.015	21446656	23	9
24	OK	0.046	23130112	32	30
25	OK	0.031	23126016	32	30
26	OK	0.031	23130112	37	35
27	OK	0.046	23134208	51	35
28	OK	0.031	23121920	64	34
29	OK	0.031	23121920	84	37
30	OK	0.046	23117824	84	36
31	OK	0.046	21434368	24	7
32	OK	0.046	21434368	24	7
33	OK	0.046	21438464	24	7
34	OK	0.312	36139008	24	24
35	OK	0.046	21442560	24	7
36	OK	0.062	21442560	24	9
37	OK	0.046	21446656	24	9
38	OK	0.328	36110336	33	16
39	OK	0.328	36114432	33	30
40	OK	0.343	36134912	38	35
41	OK	0.328	36143104	52	35
42	OK	0.343	36200448	66	35
43	OK	0.328	36114432	84	37
44	OK	0.343	36139008	85	37
45	OK	0.312	21422080	25	7
46	OK	0.312	21438464	25	7
47	OK	0.312	21434368	25	7
48	OK	3.843	174858240	25	27
49	OK	0.296	21442560	25	7
50	OK	0.312	21442560	25	9

51	OK	0.312	21438464	25	9
52	OK	1.968	36110336	34	16
53	OK	3.890	175190016	34	30
54	OK	3.953	176107520	39	35
55	OK	3.953	174637056	51	35
56	OK	3.953	174338048	67	35
57	OK	3.984	175140864	87	37
58	OK	4.000	174944256	87	37
59	OK	4.000	175235072	87	35
60	OK	3.968	174563328	86	37
61	OK	3.968	174424064	87	37
62	OK	3.968	174665728	86	37
63	OK	4.000	175329280	86	37
64	OK	3.968	173940736	86	37
65	OK	3.968	175345664	87	37
66	OK	3.968	175616000	85	35
67	OK	3.968	174804992	85	36
68	OK	3.984	174964736	87	36