КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ТАРАСА ШЕВЧЕНКА

**ЗВІТ ДО ЛАБОРАТОРНОЇ РОБОТИ**

**Пядика Любомира Сергійовича**

студента групи МІ-3, факультету кібернетики КНУ

Київ – 2018

# Постановка задачі

Реалізувати структуру данних типу "множина рядків".

Рядки непусті послідовності довжиною до 15 символів рядкових латинських літер.

Структура данних повинна підтримувати операції додавання рідку до множини, перевіри належності, вилучення.

Максимальна кількість елементів у множині, що зберігається - 106.

"Ідеальна асимптотика" O(m+n), насправді - O(mn) за рахунок колізій

Вх.данні

Кожен рядок вхідних данних задає одну операцію над множиною.

Запис операції - тип операції та наступний за ним рядку над яким проводить операцію.

+ - insert

? - check

- - delete

К-сть операцій у вх.файлі до 106.

Список операцій завершується рядком із символом #.

При додавання елемента не гарантується, що він відсутній у множині.

При вилученні не гарантується присутність.

Вих.данні

Для кожної операції ? - рядок Yes/No.

Для віднімання - повідомлення про успішність операції.

# Алгоритм та його аналіз

В основі алгоритму лежить хеш-таблиця, яка забезпечує необхідну складність виконання кожної з операцій. У даній задачі було створено таблицю розміром на 1000 елементів, і для кожного вхідного рядку обчислюється значення хеш-функції, яке визначає у який зв’язний список рядок попаде. Якщо виникатимуть колізії, зв’язні списки міститимуть більше одного елемента. Розглянемо окремо кожну операцію із структурою даних:

* Вставка (insert). Обчислюється значення хеш-функції, береться відповідний список і додається до списку. Складність обчислення хеш-функції O(1), тому кінцева складність така ж сама.

void insert(const string& s)

{

auto& list = m\_data[hash(s)];

list.push\_back(s);

}

* Перевірка на наявність (check). Використовується хеш-функція і відбувається прохід по звязному списку. У середньому колізій не повинно бути, тому складність виконання операції у середньому O(1), але у гіршому випадку O(N).

bool contains(const string& s)

{

auto& list = m\_data[hash(s)];

return find(list.begin(), list.end(), s) != list.end();

}

* Вилучення (delete). Дана операція подібна до попередньої, окрім тільки вилучення елемента із списку, яке виконується за константний час. Таким чином, складність також O(1) у середньому.

bool erase(const string& s)

{

auto& list = m\_data[hash(s)];

auto it = find(list.begin(), list.end(), s);

if (it != list.end())

{

list.erase(it);

return true;

}

return false;

}

Отже, даний алгоритм у середньому виконується за час O(N+M), де M – кількість виконуваних операцій вставки, вилучення чи перевірки на наявність.

# Лістинг програми

class string\_set

{

public:

string\_set(size\_t capacity)

: m\_data(capacity)

{}

void insert(const string& s)

{

auto& list = m\_data[hash(s)];

list.push\_back(s);

}

bool contains(const string& s)

{

auto& list = m\_data[hash(s)];

return find(list.begin(), list.end(), s) != list.end();

}

bool erase(const string& s)

{

auto& list = m\_data[hash(s)];

auto it = find(list.begin(), list.end(), s);

if (it != list.end())

{

list.erase(it);

return true;

}

return false;

}

private:

vector<list<string>> m\_data;

constexpr static int MOD = 1e9 + 7;

int hash(const string& s)

{

int result = 0;

for (size\_t i = 0; i < s.size(); ++i)

{

result = (result \* 2 + s[i]) % MOD;

}

return result % m\_data.size();

}

};

int main()

{

string\_set s(1000);

ifstream file("input.txt");

string type;

while (file >> type)

{

string value;

if (type == "+")

{

file >> value;

s.insert(value);

}

else if (type == "?")

{

file >> value;

cout << "Contains '" << value << "'? = " << (s.contains(value) ? "OK" : "FAIL") << endl;

}

else if (type == "-")

{

file >> value;

cout << "Erase '" << value << "' = " << (s.erase(value) ? "OK" : "FAIL") << endl;

}

else

{

break;

}

}

file.close();

return 0;

}