Об’єктно орієнтоване програмування

**Лабораторна робота №7**

# Хеш-таблиці

# Теоретичні відомості

Хеш-таблиця – це структура даних, що реалізує інтерфейс асоціативного масиву, а саме, вона дозволяє зберігати пари (ключ, значення) і виконувати три операції: операцію додавання нової пари, операцію пошуку і операцію видалення пари по ключу.

Існує два основні варіанти хеш-таблиць: з ланцюжками і відкритою адресацією. Хеш-таблиця містить деякий масив *H*, елементи якого є пари (хеш-таблиця з відкритою адресацією) або списки пар (хеш-таблиця з ланцюжками).

Будь-яка операція в хеш-таблиці починається з обчислення хеш-функції від ключа. Хеш-значення *i* = hash(key) грає роль індексу в масиві *H*. Потім виконується операція додавання, видалення або пошуку. Під час цієї операції відбувається доступ до об'єкту, що зберігається у відповідній комірці масиву *H*[*i*].

Ситуація, коли для різних ключів виходить один і той же хеш-значення, називається колізією. Такі події не так вже й рідкісні – наприклад, при вставці в хеш-таблицю розміром 365 осередків всього лише 23-х елементів ймовірність колізії вже перевищить 50% (якщо кожен елемент може рівноімовірно потрапити в будь-яку клітинку). Тому механізм вирішення колізій – це важлива складова будь-якої хеш-таблиці.

У деяких спеціальних випадках вдається уникнути колізій взагалі. Наприклад, якщо всі ключі елементів відомі заздалегідь (або дуже рідко змінюються), то для них можна знайти деяку досконалу хеш-функцію, яка розподілить їх по осередках хеш-таблиці без колізій. Хеш-таблиці, що використовують подібні хеш-функції, не потребують механізмі вирішення колізій, і називаються хеш-таблицями з прямою адресацією.

Число збережених елементів, поділене на розмір масиву H (число можливих значень хеш-функції), називається коефіцієнтом заповнення хеш-таблиці (load factor) і є важливим параметром, від якого залежить середній час виконання операцій.

Важлива властивість хеш-таблиць полягає в тому, що, при деяких розумних припущеннях, все три операції (пошук, вставка, видалення елементів) в середньому виконуються за час O(1). Але при цьому не гарантується, що час виконання окремої операції малий. Це пов'язано з тим, що при досягненні деякого значення коефіцієнта заповнення необхідно здійснювати перебудову індексу хеш-таблиці: збільшити значення розміру масиву *H* і заново додати в порожню хеш-таблицю всі пари.

**Відкрита адресація.** У масиві *H* зберігаються самі пари ключ-значення. Алгоритм вставки елемента перевіряє осередку масиву *H* в деякому порядку до тих пір, поки не буде знайдена перша вільна комірка, в яку і буде записаний новий елемент. Цей порядок обчислюється на льоту, що дозволяє заощадити на пам'яті для покажчиків, що вимагаються в хеш-таблицях з ланцюжками.

Послідовність, в якій проглядаються комірки хеш-таблиці, називається послідовністю проб. У загальному випадку, вона залежить тільки від ключа елемента, тобто це послідовність *h*0(*x*), *h*1(*x*), …, *h*n-1(*x*), де *x* – ключ елемента, а *hi*(*x*) – довільні функції, що зіставляють кожному ключу осередок в хеш-таблиці. Перший елемент в послідовності, як правило, дорівнює значенню деякої хеш-функції від ключа, а решта вважаються від нього одним з наведених нижче способів. Для успішної роботи алгоритмів пошуку послідовність проб повинна бути такою, щоб усі осередки хеш-таблиці опинилися переглянутих рівно по одному разу.

Алгоритм пошуку переглядає комірки хеш-таблиці в тому ж самому порядку, що і при вставці, до тих пір, поки не знайдеться або елемент з шуканим ключем, або вільна комірка (це означає відсутність елемента в хеш-таблиці).



Рисунок 7.1 – Приклад хеш-таблиці з відкритою (https://uk.wikipedia.org/wiki/Геш-таблиця).

Видалення елементів в такій схемі кілька утруднено. Зазвичай у такому випадку заводять булевий прапор для кожної комірки, і позначають, вилучений елемент в ній чи ні. Тоді видалення елемента полягає в установці цього прапора для відповідного осередку хеш-таблиці, але при цьому необхідно модифікувати процедуру пошуку існуючого елемента так, щоб вона вважала вилучені осередки зайнятими, а процедуру оновлення так, щоб вона їх вважала вільними і скидала значення прапора при додаванні.

**Приклад.** Написати програму, яка хеш таблицю для зберігання пар (рядок, ціле число). Вміст хеш таблиці зв’язується з *QTableWidget* таким чином, щоб користувач міг бачити актуальний стан хеш-таблиці (рис. 7.2).

Лістинг 6.1 – Файл hash\_table.h

#ifndef HASH\_TABLE\_H

#define HASH\_TABLE\_H

#include <QTableWidget>

struct **hash\_table\_entry**

{

QString k;

int v;

**hash\_table\_entry**(QString k, int v)

{

this->k = k;

this->v = v;

}

};

class **hash\_table**

{

private:

const unsigned int T\_S = 13;

hash\_table\_entry \*\*t;

QTableWidget \*qtable; // QTableWidget для відображення

public:

**hash\_table**();

~**hash\_table**();

int **HashFunc**(QString k);

void **Insert**(QString k, int v);

int **SearchKey**(QString k);

void **Remove**(QString k);

void **setQTable**(QTableWidget\* qtable);

void **PrintToQTable**();

};

#endif // HASH\_TABLE\_H

Лістинг 6.2 – Файл hash\_table.cpp

#include "hash\_table.h"

#include <QTableWidget>

hash\_table::**hash\_table**()

{

qtable = nullptr;

t = new hash\_table\_entry \* [T\_S];

for (unsigned int i = 0; i < T\_S; i++)

{

t[i] = nullptr;

}

}

hash\_table::~**hash\_table**()

{

for (unsigned int i = 0; i < T\_S; i++)

{

if (t[i] != nullptr)

delete t[i];

}

delete[] t;

}

int hash\_table::**HashFunc**(QString k)

{

unsigned int s = 0;

for (int i =0; i< k.length(); i++)

s += k[i].unicode();

return int(s % T\_S);

}

void hash\_table::**Insert**(QString k, int v)

{

int h = HashFunc(k);

while (t[h] != nullptr && t[h]->k != k)

{

h = HashFunc(k + " ");

}

if (t[h] != nullptr)

delete t[h];

t[h] = new hash\_table\_entry(k, v);

PrintToQTable();

}

int hash\_table::**SearchKey**(QString k)

{

int h = HashFunc(k);

while (t[h] != nullptr && t[h]->k != k)

{

h = HashFunc(k + " ");

}

if (t[h] == nullptr)

return -1;

else

return t[h]->v;

}

void hash\_table::**Remove**(QString k)

{

int h = HashFunc(k);

while (t[h] != nullptr) {

if (t[h]->k == k)

break;

h = HashFunc(k + " ");

}

if (t[h] == nullptr) {

// cout<<"No Element found at key "<<k<<endl;

return;

} else {

delete t[h];

}

PrintToQTable();

}

void hash\_table::**setQTable**(QTableWidget\* qtable)

{

this->qtable = qtable;

}

void hash\_table::**PrintToQTable**()

{

qtable->setRowCount(int(T\_S));

for (int i = 0; i < int(T\_S); i++)

{

if (t[i])

{

qtable->setItem(i, 0, new QTableWidgetItem(t[i]->k));

qtable->setItem(i, 1, new QTableWidgetItem(QString::number(t[i]->v)));

}

}

}

Лістинг 6.3 – Файл mainwindow.cpp

#include "mainwindow.h"

#include "ui\_mainwindow.h"

#include "hash\_table.h"

hash\_table ht;

MainWindow::**MainWindow**(QWidget \*parent) :

QMainWindow(parent),

ui(new Ui::MainWindow)

{

ui->setupUi(this);

// Налаштування

ht.setQTable(ui->tableWidget);

ui->tableWidget->setColumnCount(2);

ui->tableWidget->setHorizontalHeaderItem(0, new QTableWidgetItem("Ключ"));

ui->tableWidget->setHorizontalHeaderItem(1, new QTableWidgetItem("Значення"));

}

MainWindow::~***MainWindow***()

{

delete ui;

}

void MainWindow::**on\_pushButton\_clicked**()

{

ht.Insert(ui->lineEdit->text(),ui->lineEdit\_2->text().toInt());

}

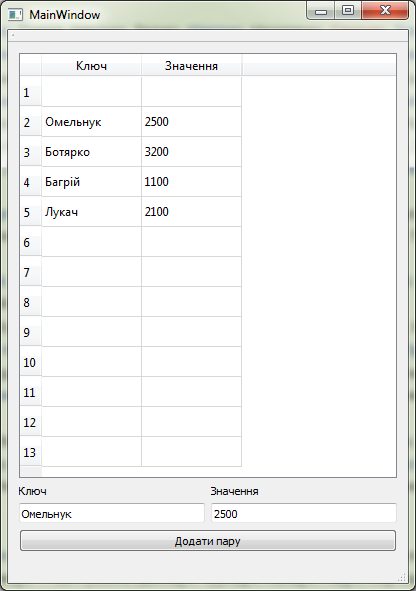


Рисунок 7.2 – Результат роботи програми

**Лабораторна робота №7 (3-й семестр)**

Написати програму, в якій

* 1. Реалізовано клас, що описує хеш-таблицю для зберігання пар (рядок, дійсне число);
  2. Користувач має можливість додавати, видаляти або знаходити за ключем відповідні пари;
  3. Вміст хеш-таблиці зв'язується з деяким компонентом таким чином, щоб користувач міг бачити його актуальний стан. Користувачеві також виводиться результат виконання п. 4, 5, 6;
  4. У класі реалізований оператор «+» для додавання дійсного числа до всіх елементів хеш-таблиці;
  5. У класі реалізовано оператор **[ ]** для отримання значення за ключем;
  6. Користувачу виводиться значення коефіцієнту заповнення за допомогою QProgressBar.

Написати програму, в якій

* 1. Реалізовано клас, що описує хеш-таблицю для зберігання пар (рядок, дійсне число);
  2. Користувач має можливість додавати, видаляти або знаходити за ключем відповідні пари;
  3. Вміст хеш-таблиці зв'язується з деяким компонентом таким чином, щоб користувач міг бачити його актуальний стан. Користувачеві також виводиться результат виконання п. 4, 5, 6;
  4. У класі реалізований оператор «–» для віднімання дійсного числа від всіх елементів хеш-таблиці;
  5. У класі реалізовано оператор **[ ]** для отримання значення за ключем;
  6. Користувачу виводиться значення коефіцієнту заповнення за допомогою QProgressBar.

Написати програму, в якій

* 1. Реалізовано клас, що описує хеш-таблицю для зберігання пар (рядок, дійсне число);
  2. Користувач має можливість додавати, видаляти або знаходити за ключем відповідні пари;
  3. Вміст хеш-таблиці зв'язується з деяким компонентом таким чином, щоб користувач міг бачити його актуальний стан. Користувачеві також виводиться результат виконання п. 4, 5, 6;
  4. У класі реалізований оператор «\*» для множення дійсного числа на всі елементи хеш-таблиці;
  5. У класі реалізовано оператор **[ ]** для отримання значення за ключем;
  6. Користувачу виводиться значення коефіцієнту заповнення за допомогою QProgressBar.

Написати програму, в якій

* 1. Реалізовано клас, що описує хеш-таблицю для зберігання пар (рядок, дійсне число);
  2. Користувач має можливість додавати, видаляти або знаходити за ключем відповідні пари;
  3. Вміст хеш-таблиці зв'язується з деяким компонентом таким чином, щоб користувач міг бачити його актуальний стан. Користувачеві також виводиться результат виконання п. 4, 5, 6;
  4. У класі реалізований оператор «/» для ділення всіх елементів хеш-таблиці на дійсне число;
  5. У класі реалізовано оператор **[ ]** для отримання значення за ключем;
  6. Користувачу виводиться значення коефіцієнту заповнення за допомогою QProgressBar.

Написати програму, в якій

* 1. Реалізовано клас, що описує хеш-таблицю для зберігання пар (рядок, дійсне число);
  2. Користувач має можливість додавати, видаляти або знаходити за ключем відповідні пари;
  3. Вміст хеш-таблиці зв'язується з деяким компонентом таким чином, щоб користувач міг бачити його актуальний стан. Користувачеві також виводиться результат виконання п. 4, 5, 6;
  4. У класі реалізований оператор «++» для додавання одиниці до всіх елементів хеш-таблиці;
  5. У класі реалізовано оператор **[ ]** для отримання значення за ключем;
  6. Користувачу виводиться значення коефіцієнту заповнення за допомогою QProgressBar.

Написати програму, в якій

* 1. Реалізовано клас, що описує хеш-таблицю для зберігання пар (рядок, дійсне число);
  2. Користувач має можливість додавати, видаляти або знаходити за ключем відповідні пари;
  3. Вміст хеш-таблиці зв'язується з деяким компонентом таким чином, щоб користувач міг бачити його актуальний стан. Користувачеві також виводиться результат виконання п. 4, 5, 6;
  4. У класі реалізований оператор «– –» для віднімання одиниці від всіх елементів хеш-таблиці;
  5. У класі реалізовано оператор **[ ]** для отримання значення за ключем;
  6. Користувачу виводиться значення коефіцієнту заповнення за допомогою QProgressBar.

Написати програму, в якій

* 1. Реалізовано клас, що описує хеш-таблицю для зберігання пар (рядок, дійсне число);
  2. Користувач має можливість додавати, видаляти або знаходити за ключем відповідні пари;
  3. Вміст хеш-таблиці зв'язується з деяким компонентом таким чином, щоб користувач міг бачити його актуальний стан. Користувачеві також виводиться результат виконання п. 4, 5, 6;
  4. У класі реалізований оператор «>», що повертає булеве значення true, коли хоч одне значення менше ніж вказане дійсне число;
  5. У класі реалізовано оператор **[ ]** для отримання значення за ключем;
  6. Користувачу виводиться значення коефіцієнту заповнення за допомогою QProgressBar.

Написати програму, в якій

* 1. Реалізовано клас, що описує хеш-таблицю для зберігання пар (рядок, дійсне число);
  2. Користувач має можливість додавати, видаляти або знаходити за ключем відповідні пари;
  3. Вміст хеш-таблиці зв'язується з деяким компонентом таким чином, щоб користувач міг бачити його актуальний стан. Користувачеві також виводиться результат виконання п. 4, 5, 6;
  4. У класі реалізований оператор «<», що повертає булеве значення true, коли хоч одне значення більше ніж вказане дійсне число;
  5. У класі реалізовано оператор **[ ]** для отримання значення за ключем;
  6. Користувачу виводиться значення коефіцієнту заповнення за допомогою QProgressBar.

Написати програму, в якій

* 1. Реалізовано клас, що описує хеш-таблицю для зберігання пар (рядок, дійсне число);
  2. Користувач має можливість додавати, видаляти або знаходити за ключем відповідні пари;
  3. Вміст хеш-таблиці зв'язується з деяким компонентом таким чином, щоб користувач міг бачити його актуальний стан. Користувачеві також виводиться результат виконання п. 4, 5, 6;
  4. У класі реалізований оператор «==», що повертає булеве значення true, коли хоч одне значення дорівнює вказаному дійсному числу;
  5. У класі реалізовано оператор **[ ]** для отримання значення за ключем;
  6. Користувачу виводиться значення коефіцієнту заповнення за допомогою QProgressBar.