МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования
«МИРЭА – Российский технологический университет»

РТУ МИРЭА

Институт информационных технологий (ИТ)
Кафедра математического обеспечения и стандартизации информационных технологий (МОСИТ)

ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ № 6.1

по дисциплине
«Структуры и алгоритмы обработки данных»

Тема: «Быстрый доступ к данным с помощью хеш-таблиц»

Выполнил студент: Лазаренко С.А.
Группа: ИКБО-10-23

Москва 2024

СОДЕРЖАНИЕ

ЦЕЛЬ РАБОТЫ

Освоить приёмы хеширования и эффективного поиска элементов множества. Разработать программу для управления динамическим множеством данных с использованием хеш-таблицы, позволяющей осуществлять быстрый прямой доступ к элементам по ключу. Основной задачей является реализация базовых операций работы с множеством данных (вставка, удаление, поиск и вывод) с использованием хеш-таблицы и выбранной структуры данных, описанной в индивидуальном варианте. Программа должна быть организована в виде класса, который содержит массив данных и хеш-таблицу, а также реализует механизм автоматического расширения и рехеширования при необходимости.

Хеш-функция и метод разрешения коллизий подбираются самостоятельно, с соблюдением общепринятых правил их выбора. В процессе работы предусмотрена автоматическая инициализация хеш-таблицы с пятью-семью записями, а также возможность работы с таблицей через текстовый командный интерфейс, позволяющий пользователю выполнять основные операции в произвольной последовательности. Программа должна быть протестирована на корректность выполнения всех базовых операций, включая обработку коллизий и автоматическое рехеширование, чтобы обеспечить эффективное и надежное функционирование хеш-таблицы.

ХОД РАБОТЫ

Формулировка задачи

Разработать программу, которое использует хеш-таблицу для организации прямого доступа к элементам динамического множества полезных данных. Множество реализуйте на массиве, структура элементов (перечень полей) которого приведена в индивидуальном варианте.

Программа должна содержать класс с базовыми операциями: вставки, удаления, поиска по ключу, вывода. Включить в класс массив полезных данных и хеш-таблицу. Хеш-функцию подобрать самостоятельно, используя правила выбора функции.

Реализовать расширение размера таблицы и рехеширование, когда это требуется, в соответствии с типом разрешения коллизий.

Предусмотреть автоматическое заполнение таблицы 5-7 записями.

Реализовать текстовый командный интерфейс пользователя для возможности вызова методов в любой произвольной последовательности, сопроводите вывод достаточными для понимания происходящего сторонним пользователем подсказками.

Провести полное тестирование программы. Результаты тестирования включите в отчет по выполненной работе.

Индивидуальный вариант работы - 2

Описание подхода к решению

Подход к решению задачи с использованием хеш-таблицы базируется на создании эффективной структуры данных для быстрого доступа к банковским счетам. В основе решения лежит применение метода цепного хеширования, позволяющего обрабатывать ситуации, когда несколько записей имеют одинаковый хеш. Для хранения данных о банковских счетах используется структура Data_Record, которая содержит номер счета, ФИО владельца и адрес, а также указатель на следующий элемент, что позволяет организовать связные списки при возникновении коллизий.

Основная функциональность реализована в классе Hash_Table, который управляет массивом указателей на объекты Data_Record. Для вычисления индекса в массиве используется хеш-функция, рассчитывающая остаток от деления номера счета на размер таблицы. Эта функция обеспечивает равномерное распределение записей, минимизируя коллизии. Вставка новых записей происходит в начало связного списка по соответствующему индексу, и при достижении определенного уровня заполнения таблица автоматически расширяется, что сохраняет эффективность операций.

При удалении и поиске программа сначала определяет индекс с помощью хеш-функции, а затем перебирает элементы в соответствующем связном списке. В случае превышения порога заполнения запускается процедура рехеширования, которая удваивает размер таблицы и перераспределяет все записи. Пользователь взаимодействует с программой через текстовый интерфейс, что позволяет добавлять, удалять, искать и выводить записи.

Коды программы

Реализуем код программы на языке программирования C++ :

```cpp
#include <iostream>
#include <string>

using namespace std;

struct Data_Record {
    int account_number;
    string full_name;
    string address;
    Data_Record *next;

    Data_Record(int account_number, const string& full_name, const string& address)
        : account_number(account_number), full_name(full_name), address(address), next(nullptr) {}

    string string_rep();
};

class Hash_Table {
private:
    Data_Record **table;
    int size;
    int num_of_elements;
    const float load_factor_threshold = 0.75;

    int hash_function(int account_number) const;
    void rehash();

public:
    Hash_Table(int table_size);
    ~Hash_Table();

    void insert(Data_Record *new_record);
    Data_Record *search(int account_number);
    int remove(int account_number);
    void display() const;
};
```

Рисунок 1 – код программы в hash.hpp

```
string Data_Record::string_rep() {
    return "[Номер счета: " + to_string(account_number) + ", ФИО: " + full_name + ", Адрес: " + address + "]";
}

Hash_Table::Hash_Table(int table_size) : size(table_size), num_of_elements(0) {
    table = new Data_Record *[size];
    for (int i = 0; i < size; i++) {
        table[i] = nullptr;
    }
}

Hash_Table::~Hash_Table() {
    for (int i = 0; i < size; i++) {
        Data_Record *current = table[i];
        while (current != nullptr) {
            Data_Record *temp = current;
            current = current->next;
            delete temp;
        }
    }
    delete[] table;
}

int Hash_Table::hash_function(int account_number) const {
    return account_number % size;
}

void Hash_Table::rehash() {
    cout << "Перехеширование: увеличиваем размер таблицы до " << size * 2 << endl;
    int old_size = size;
    size *= 2;
    Data_Record **new_table = new Data_Record *[size];
    for (int i = 0; i < size; i++) {
        new_table[i] = nullptr;
    }
```

Рисунок 2 – Файл hash.cpp (часть 1)

```
    for (int i = 0; i < old_size; i++) {
        Data_Record *current = table[i];
        while (current != nullptr) {
            Data_Record *next_record = current->next;
            int new_index = hash_function(current->account_number);
            current->next = new_table[new_index];
            new_table[new_index] = current;
            current = next_record;
        }
    }

    delete[] table;
    table = new_table;
}

void Hash_Table::insert(Data_Record *new_record) {
    int index = hash_function(new_record->account_number);
    new_record->next = table[index];
    table[index] = new_record;
    num_of_elements++;
    if (static_cast<float>(num_of_elements) / size > load_factor_threshold) {
        rehash();
    }
}

Data_Record *Hash_Table::search(int account_number) {
    int index = hash_function(account_number);
    Data_Record *current = table[index];
    while (current != nullptr) {
        if (current->account_number == account_number) {
            return current;
        }
        current = current->next;
    }
    return nullptr;
```

Рисунок 3 – Файл hash.cpp (часть 2)

```
    while (current != nullptr) {
        if (current->account_number == account_number) {
            if (prev == nullptr) {
                table[index] = current->next;
            } else {
                prev->next = current->next;
            }
            delete current;
            num_of_elements--;
            return 0;
        }
        prev = current;
        current = current->next;
    }
    return 1;
}

void Hash_Table::display() const {
    bool empty_flag = true;
    for (int i = 0; i < size; i++) {
        Data_Record *current = table[i];
        if (current != nullptr) {
            empty_flag = false;
            cout << "Ячейка " << i << ":" << endl;
            while (current != nullptr) {
                cout << "  " << current->string_rep() << endl;
                current = current->next;
            }
            cout << endl;
        }
    }
    if (empty_flag) {
        cout << "Нет данных" << endl;
    }
}
```

Рисунок 4 – Файл hash.cpp (часть 3)

```
Hash_Table bankAccounts(5);

void add_initial_records() {
    bankAccounts.insert(new Data_Record(1234567, "Лазаренко Сергей Александрович", "ул. Нарвская, д.12"));
    bankAccounts.insert(new Data_Record(2345678, "Петров Сергей Петрович", "ул. Гагарина, д.5"));
    bankAccounts.insert(new Data_Record(3456789, "Сидоров Сидор Сидорович", "ул. Мира, д.10"));
    bankAccounts.insert(new Data_Record(4567890, "Николаев Николай Николаевич", "ул. Победы, д.2"));
    bankAccounts.insert(new Data_Record(5678901, "Федоров Федор Федорович", "ул. Парковая, д.3"));
}

void clear_cin() {
    cin.clear();
    cin.ignore(numeric_limits<streamsize>::max(), '\n');
}

int get_account_number_from_user() {
    int account_number;
    while (true) {
        cout << "Введите номер счета (7-значное число): ";
        if (cin >> account_number && account_number >= 1000000 && account_number <= 9999999) {
            clear_cin();
            return account_number;
        }
        clear_cin();
        cout << "Неверный номер счета. Пожалуйста, введите 7-значное число.\n";
    }
}

string get_string_from_user(string prompt) {
    string s;
    cout << prompt;
    getline(cin, s);
    return s;
}
```

Рисунок 5 – Файл main.cpp (часть 1)

```
void insert() {
    int account_number = get_account_number_from_user();
    string name = get_string_from_user("Введите ФИО: ");
    string address = get_string_from_user("Введите адрес: ");
    Data_Record* new_record = new Data_Record(account_number, name, address);
    bankAccounts.insert(new_record);
    cout << "Успешно добавлен новый счет: " << new_record->string_rep() << endl;
}

void del() {
    int account_number = get_account_number_from_user();
    int return_code = bankAccounts.remove(account_number);
    if (return_code == 0) {
        cout << "Счет с номером " << account_number << " успешно удален." << endl;
    } else {
        cout << "Счет с номером " << account_number << " не найден." << endl;
    }
}

void search() {
    int account_number = get_account_number_from_user();
    Data_Record* record = bankAccounts.search(account_number);
    if (record == nullptr) {
        cout << "Счет с номером " << account_number << " не найден." << endl;
    } else {
        cout << "Найден счет: " << record->string_rep() << endl;
    }
}

void display() {
    cout << "Отображение всех счетов: " << endl;
    bankAccounts.display();
}
```

Рисунок 6 – Файл main.cpp (часть 2)

```
int main() {
    setlocale(LC_ALL, "ru_RU.UTF-8");

    add_initial_records();

    int cmd;
    while (true) {
        while (true) {
            cout << "1: Добавить новый счет" << endl;
            cout << "2: Удалить счет" << endl;
            cout << "3: Поиск счета" << endl;
            cout << "4: Отобразить все счета" << endl;
            cout << "0: Выйти" << endl;
            cout << "\tВаш выбор: ";
            if (cin >> cmd && cmd >= 0 && cmd <= 4) break;
            clear_cin();
            cout << "Неверный код команды.\n";
        }

        switch (cmd) {
            case 0:
                return 0;
            case 1:
                insert();
                break;
            case 2:
                del();
                break;
            case 3:
                search();
                break;
            case 4:
                display();
                break;
        }
    }
}
```

Рисунок 7 – Файл main.cpp (часть 3)

Результаты тестирования

Выполним тестирование программы:

```
Перехеширование: увеличиваем размер таблицы до 10
1: Добавить новый счет
2: Удалить счет
3: Поиск счета
4: Отобразить все счета
0: Выйти
    Ваш выбор: 1
Введите номер счета (7-значное число): 1238291
Введите ФИО: А Б В
Введите адрес: У
Успешно добавлен новый счет: [Номер счета: 1238291, ФИО: А Б В, Адрес: У]
1: Добавить новый счет
2: Удалить счет
3: Поиск счета
4: Отобразить все счета
0: Выйти
    Ваш выбор: 2
Введите номер счета (7-значное число): 1238291
Счет с номером 1238291 успешно удален.
1: Добавить новый счет
2: Удалить счет
3: Поиск счета
4: Отобразить все счета
0: Выйти
    Ваш выбор: 4
Отображение всех счетов:
Ячейка 0:
  [Номер счета: 4567890, ФИО: Николаев Николай Николаевич, Адрес: ул. Победы, д.2]

Ячейка 1:
  [Номер счета: 5678901, ФИО: Федоров Федор Федорович, Адрес: ул. Парковая, д.3]

Ячейка 7:
  [Номер счета: 1234567, ФИО: Лазаренко Сергей Александрович, Адрес: ул. Нарвская, д.12]

Ячейка 8:
  [Номер счета: 2345678, ФИО: Петров Сергей Петрович, Адрес: ул. Гагарина, д.5]
```

Рисунок 8 – Тестирование

Тестирование показало, что программа работает корректно.

ВЫВОД

В результате выполнения работы были освоены приёмы хеширования и эффективного поиска элементов множества. Был получен опыт реализации хэш-таблицы с цепным хешированием.

СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ

1. Страуструп Б. Программирование. Принципы и практика с использованием C++. 2-е изд., 2016.

2. Документация по языку C++ [Электронный ресурс]. URL: https://docs.microsoft.com/ru-ru/cpp/cpp/ (дата обращения 08.09.2024).

3. Курс: Структуры и алгоритмы обработки данных. Часть 2 [Электронный ресурс]. https://online-edu.mirea.ru/course/view.php?id=4020 (дата обращения 04.09.2024)