|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | |  |  |  | | МИНОБРНАУКИ РОССИИ | | | | Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** | | |   Институт Информационных технологий | |
|  | |
| Кафедра Математического обеспечения и стандартизации информационных технологий | |
|  | |
|  | |

|  |  |
| --- | --- |
| **ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ № 2.2** | |
| **по дисциплине** | |
| **«**Структуры и алгоритмы обработки данных**»**  **Тема: «Хеширование: прямой доступ к данным»** | |
|  | |
| Выполнил студент группы ИКБО-30-22 | Сенькевич Г.Д. |
| Принял преподаватель | Красников С.А. |

Москва 2023

# **Цель работы**

Освоить приёмы хеширования и эффективного поиска элементов множества.

1. **Ход работы**

# **. Формулировка задачи**

Разработайте приложение, которое использует хеш-таблицу (пары «ключ – хеш») для организации прямого доступа к элементам динамического множества полезных данных. Множество реализуйте на массиве, структура элементов (перечень полей) которого приведена в индивидуальном варианте (п.3).

Приложение должно содержать класс с базовыми операциями: вставки, удаления, поиска по ключу, вывода. Включите в класс массив полезных данных и хеш-таблицу. Хеш-функцию подберите самостоятельно, используя правила выбора функции.

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

Предусмотрите автоматическое заполнение таблицы 5-7 записями.

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

Проведите полное тестирование программы (все базовые операции, изменение размера и рехеширование), тест-примеры определите самостоятельно. Результаты тестирования включите в отчет по выполненной работе.

Вариант – 20.

Метод хеширования (тип последовательностей проб) – открытая адресация (линейное пробирование).

Структура элемента множества – книга: ISBN – двенадцатизначное число, автор, название.

* 1. **. Математическая модель решения**

Для решения задачи были использованы такие операции как битовый сдвиг и побитовые логические операции.

1. Битовый сдвиг:

* Левый битовый сдвиг (<<): Эта операция сдвигает биты числа влево на указанное количество позиций.
* Правый битовый сдвиг (>>): Эта операция сдвигает биты числа вправо на указанное количество позиций.

1. Побитовые логические операции:

* Побитовое И (&): Эта операция выполняет логическое И между каждой парой битов двух чисел. Результат равен 1, только если оба бита равны 1.
* Побитовое ИЛИ (|): Эта операция выполняет логическое ИЛИ между каждой парой битов двух чисел. Результат равен 1, если хотя бы один из битов равен 1.

С помощью этих операций программист получает возможность взаимодействовать с отдельными битами чисел, что позволяет реализовать сортировку с использованием битового массива.

Сортировка с использованием битового массива — это метод, который позволяет упорядочить набор элементов путем использования битов для отображения присутствия или отсутствия каждого элемента в сортированном массиве. Вот как это происходит:

1. **Инициализация битового массива:** создается битовый массив, который будет использоваться для отслеживания присутствия элементов в исходном массиве. Размер битового массива определяется максимальным значением элементов в исходном массиве.
2. **Установка битов:** для каждого элемента в исходном массиве происходит установка соответствующего бита в битовом массиве. Например, если элемент в исходном массиве равен 5, то бит с индексом 5 в битовом массиве устанавливается в 1, что указывает на то, что элемент присутствует в исходном массиве.
3. **Извлечение элементов:** после того как все элементы из исходного массива были обработаны и битовый массив правильно настроен, можно начать извлечение элементов в сортированном порядке. Для этого проходят по битовому массиву и извлекают элементы, соответствующие установленным битам. Эти элементы извлекаются в порядке возрастания индексов битов.
4. **Сортированный массив:** получается отсортированный массив, в котором элементы упорядочены по возрастанию.
   1. **. Код программы**

Заголовочный файл, описывающий классы book и book\_hashmap, первый из которых создан для того, чтобы хранить информацию о книге, а второй реализует хэш-таблицу, хранящую экземпляры класса book, приведён в листинге 1.

Листинг 1 ­— Описание классов book и book\_hashmap.

|  |
| --- |
| #ifndef BOOK\_HASHMAP\_H  #define BOOK\_HASHMAP\_H  #include <iostream>  #include <string>  typedef unsigned long long ull;  class book // 32 bytes  {  private:  ull isbn;  std::string author;  std::string name;  public:  book();  book(ull isbn, std::string author, std::string name);  book(const book& b);  ull get\_isbn() const;  std::string get\_author() const;  std::string get\_name() const;  friend std::ostream& operator<<(std::ostream& os, const book& b);  friend std::istream& operator>>(std::istream& is, book& b);  };  class book\_hashmap  {  private:  const size\_t initial\_size = 1000;  const float capacity = 1.5f;  size\_t size;  book\*\* books;  size\_t hash(ull isbn) const;  size\_t hash(const book& b) const;  void resize();  public:  book\_hashmap();  size\_t hashmap\_size() const;  book\* get(size\_t index) const;  int search(ull isbn) const;  size\_t insert(const book& b);  bool remove\_by\_isbn(ull isbn);  bool remove\_by\_index(size\_t index);  friend std::ostream& operator<<(std::ostream& os, const book\_hashmap& bh);  };  #endif |

Реализация операций поиска по ключу, вставки и удаления приведена в листинге 2.

Листинг 2 ­— Реализация базовых операций.

|  |
| --- |
| int book\_hashmap::search(ull isbn) const  {  size\_t index = hash(isbn);  for (; index < size; ++index)  {  if (books[index] != nullptr && books[index]->get\_isbn() == isbn)  {  return index;  }  }  return -1;  }  size\_t book\_hashmap::insert(const book& b)  {  size\_t index = hash(b);  while (books[index] != nullptr)  {  ++index;  if (index >= size)  {  resize();  return insert(b);  }  }  books[index] = new book(b);  return index;  }  bool book\_hashmap::remove\_by\_isbn(ull isbn)  {  int index = search(isbn);  if (index == -1)  {  return false;  }  else  {  return remove\_by\_index(index);  }  }  bool book\_hashmap::remove\_by\_index(size\_t index)  {  if (index >= size || books[index] == nullptr)  {  return false;  }  books[index] = nullptr;  return true;  } |

Функция, расширяющая размер таблицы и вместе с тем осуществляющая рехэширование, представлена в листинге 3.

Листинг 3 ­— Функция увеличения размера таблицы.

|  |
| --- |
| void book\_hashmap::resize()  {  size\_t old\_size = size;  book\*\* old\_books = new book\*[old\_size];  for (size\_t i = 0; i < old\_size; ++i)  {  if (books[i] != nullptr)  {  old\_books[i] = new book(\*books[i]);  }  else  {  old\_books[i] = nullptr;  }  }  delete[] books;  size = (size\_t)(size \* capacity);  books = new book\*[size];  for (size\_t i = 0; i < size; ++i)  {  books[i] = nullptr;  }  for (size\_t i = 0; i < old\_size; ++i)  {  if (old\_books[i] != nullptr)  {  insert(\*old\_books[i]);  }  }  delete[] old\_books;  } |

Функция, заполняющая таблицу заранее определёнными тестовыми записями для последующего взаимодействия с ней, представлена в листинге 4.

Листинг 4 ­— Функция, заполняющая таблицу тестовыми записями.

|  |
| --- |
| void fill\_hashmap\_sample\_data(book\_hashmap\* books)  {  std::vector<std::tuple<ull, std::string, std::string>> data = {  std::make\_tuple(978723638028, "Tolstoi", "Anna Karenina"),  std::make\_tuple(978960292031, "Dostoevsky", "Crime and Punishment"),  std::make\_tuple(978228419636, "Pelevin", "Chapayev and Void"),  std::make\_tuple(978519673587, "Yerofeyev", "Moscow-Petushki"),  std::make\_tuple(978968135006, "Bulgakov", "The Master and Margarita"),  std::make\_tuple(978581156609, "Gogol", "Dead Souls")  };  for (auto entry : data)  {  book b = book(std::get<0>(entry), std::get<1>(entry), std::get<2>(entry));  books->insert(b);  }  } |

Функция, запускающая командный интерфейс пользователя, позволяющий ему выполнять все базовые операции с таблицей, а также заполнить её тестовыми записями с помощью вышеприведённой функции, представлена в листинге 5.

Листинг 5 ­— Функция, реализующая командный интерфейс пользователя.

|  |
| --- |
| void run\_shell()  {  book\_hashmap\* books = new book\_hashmap();    std::cout << "========================================\n"  << "1 - Заполнить таблицу тестовыми записями\n"  << "2 - Продолжить без заполнения\n";  int fill\_hashmap\_choice = 0;  std::cin >> fill\_hashmap\_choice;  switch (fill\_hashmap\_choice)  {  case 1:  fill\_hashmap\_sample\_data(books);  break;  case 2:  break;  default:  std::cout << "Неккоректный ввод!\n";  return;  }  while (true)  {  std::cout << "========================================\n"  << "1 - Вывести таблицу\n"  << "2 - Вставить запись\n"  << "3 - Найти запись по isbn\n"  << "4 - Удалить запись по isbn\n"  << "5 - Удалить запись по индексу в таблице\n"  << "6 - Выход\n";  int action\_choice = 0;  std::cin >> action\_choice;  std::cout << "========================================\n";  if (action\_choice == 1)  {  std::cout << (\*books);  }  else if (action\_choice == 2)  {  std::cout << "Введите isbn, автора книги и её название (через пробел)\n";  book b;  std::cin >> b;  size\_t index = books->insert(b);  std::cout << "Книга была вставлена по индексу " << index << '\n';  }  else if (action\_choice == 3)  {  std::cout << "Введите isbn: ";  ull isbn = 0;  std::cin >> isbn;  int index = books->search(isbn);  if (index == -1)  {  std::cout << "Книга не найдена\n";  }  else  {  std::cout << (\*books->get(index)) << std::endl;  }  }  else if (action\_choice == 4)  {  std::cout << "Введите isbn: ";  ull isbn = 0;  std::cin >> isbn;  bool is\_deleted = books->remove\_by\_isbn(isbn);  if (is\_deleted)  {  std::cout << "Книга была успешно удалена\n";  }  else  {  std::cout << "Книга не была удалена (не найдена)\n";  }  }  else if (action\_choice == 5)  {  std::cout << "Введите индекс: ";  size\_t index = 0;  std::cin >> index;  bool is\_deleted = books->remove\_by\_index(index);  if (is\_deleted)  {  std::cout << "Книга была успешно удалена\n";  }  else  {  std::cout << "Книга не была удалена (не найдена)\n";  }  }  else if (action\_choice == 6)  {  return;  }  else  {  std::cout << "Неккоректный ввод!\n";  return;  }  }  } |

* 1. **. Результаты тестирования**

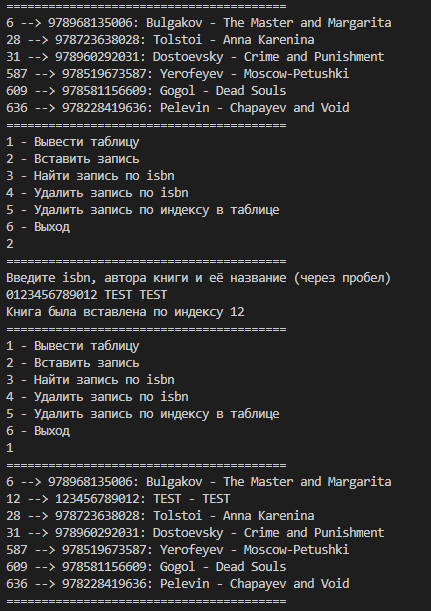
Результаты тестирования класса book\_hashmap представлены на рисунках 1 (тестирование функции вставки), 2 (тестирование функции поиска), 3, 4 (тестирование функции удаления) и 5 (автоматическое тестирование).

Рисунок 1 — Тестирование класса book\_hashmap.

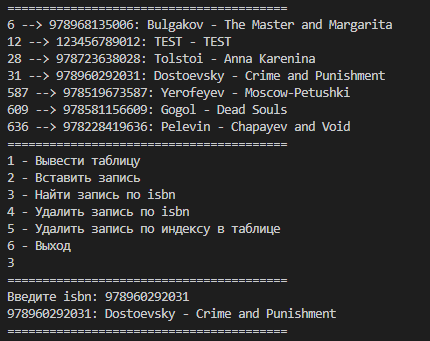


Рисунок 2 — Тестирование класса book\_hashmap.

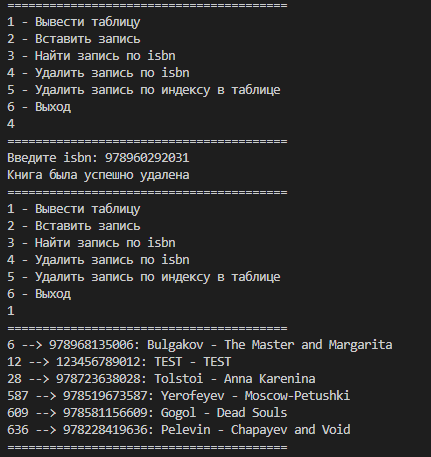
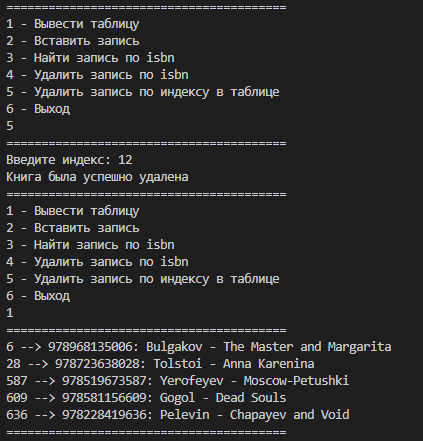


Рисунок 3 — Тестирование класса book\_hashmap.

Рисунок 4 — Тестирование класса book\_hashmap.

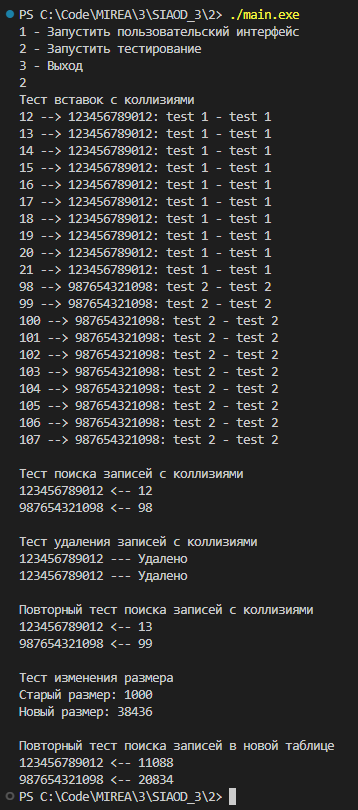


Рисунок 5 — Тестирование класса book\_hashmap.

Тестирование показало, что все программы работают правильно, корректно решая поставленные задачи.

1. **Выводы**

В результате выполнения работы я:

1. Освоил приёмы хеширования и эффективного поиска элементов множества;
2. Реализовал хэш-таблицу для хранения информации о книгах на языке программирования C++.
3. **Список литературы**
4. Страуструп Б. Программирование. Принципы и практика с использованием C++. 2-е изд., 2016.
5. Документация по языку С++ [Электронный ресурс]. URL: https://docs.microsoft.com/ruru/cpp/cpp/ (дата обращения 01.09.2021).
6. Курс: Структуры и алгоритмы обработки данных. Часть 2 [Электронный ресурс]. URL: https://online-edu.mirea.ru/course/view.php?id=4020 (дата обращения 01.09.2021).